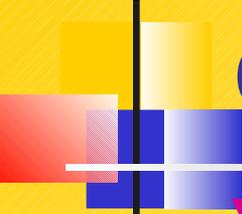


Entorno en Java y Diseño orientado a objetos

ELO-329 Diseño y programación
orientados a objetos

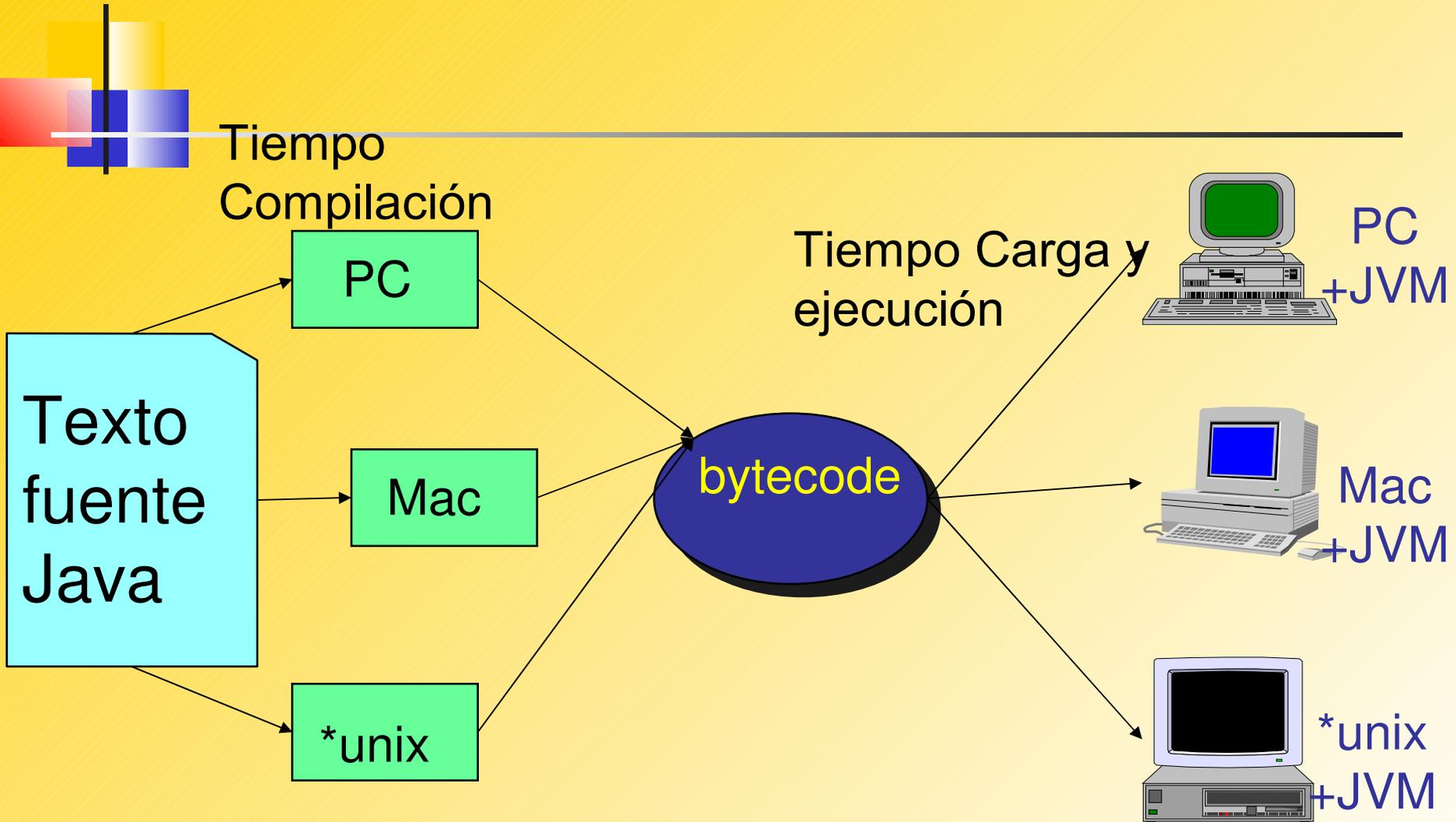
1s09

Java: Motivaciones de su origen



- **“Deja” atrás características problemáticas:**
 - Punteros
 - Asignación de memoria (malloc)
 - Herencia múltiple
 - Sobrecarga de operadores
- **Independiente de:**
 - Tipo de computador
 - Sistema operativo
 - Sistema de ventanas (win32, Motif, etc...)

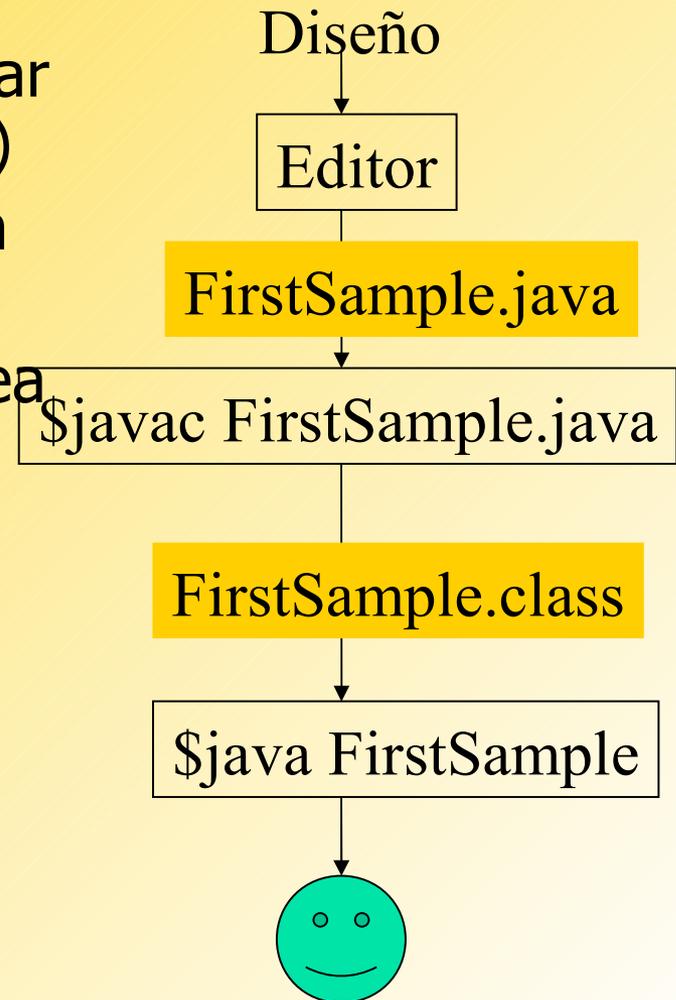
Compilación

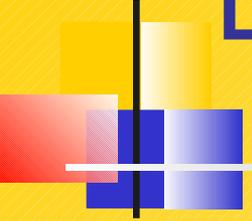


JVM es la **J**ava **V**irtual **M**achine,
Una para cada plataforma.

Trabajando con Java

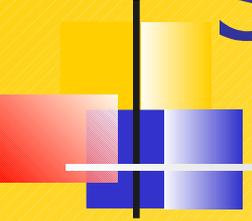
- Creación programa: Con editor crear programa *.java (FirstSample.java)
 - Hacer uso de documentación en manuales.elo.utfsm.cl
- Compilación: vía el comando el línea
`$ javac FirstSample.java`
- Ejecución:
`$ java FirstSample`
- Hay ambientes de trabajo más amigables para hacer estas tareas.





Editores de texto

- Recomiendo aprender a digitar bien.
- **Emacs** (win o Linux), Kate (linux),
- Usar ambientes integrados de Desarrollo (IDE) como:
 - **Jgraps**
 - Eclipse
 - netbean
- Hay otros, ver conveniencia.
- No usar notepad o similar.



Sistema de Desarrollo

- Lo puede bajar de SUN:
- <http://java.sun.com>
- <http://www.elo.utfsm.cl/~install>
- Versiones:
 - Java EE (Enterprise Edition),
 - Java SE (Standar Edition),
 - Java ME (Micro-Edition)

¿Cómo diseñamos programas de computación?

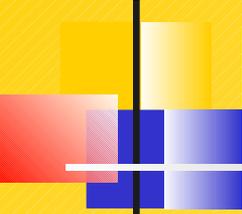


Modelado

- En todas las aplicaciones, los programadores crean modelos

Problema	Modelo	Sub-modelos
Clima	Atmósfera	Nubes, mar, viento
Obra Civil	Puente	Torres, cubierta, pilares
Contabilidad	Libro contable	Cientes registro, registro ahorros
Juego	Mundo virtual	Dragones, calabozos

- Programas modelas el comportamiento de objetos del mundo real
- Necesitamos una formalidad para crear **modelos de software** de los **objetos** que un programa maneja
- El diseño de software **orientado a objetos** usa
 - Clases** de objetos (class)
 - Métodos** que manipulan esos objetos



Diseño Orientado a Objetos

- **Clases** – Son las abstracciones del sistema.
 - Definen el comportamiento de un grupo similar de **objetos**

Clase

Puente

Comportamiento

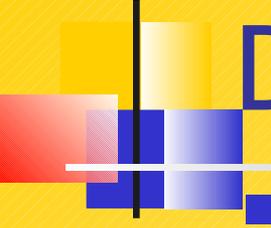
- Colapsa con vientos sobre 50km/h.
- Flexión de cubierta proporcional a la carga.

Dragon

- Puede ser creado con más de una vida.
- Si le cae un rayo de más de 4 GVolts, se encoge y transforma en un montículo de polvo de oro.

Cuenta bancaria

- Cada cuenta puede tener distinta tasa de interés.
- Sólo se permite retiros de hasta 200 K\$ diarios.



Diseño orientado a objetos

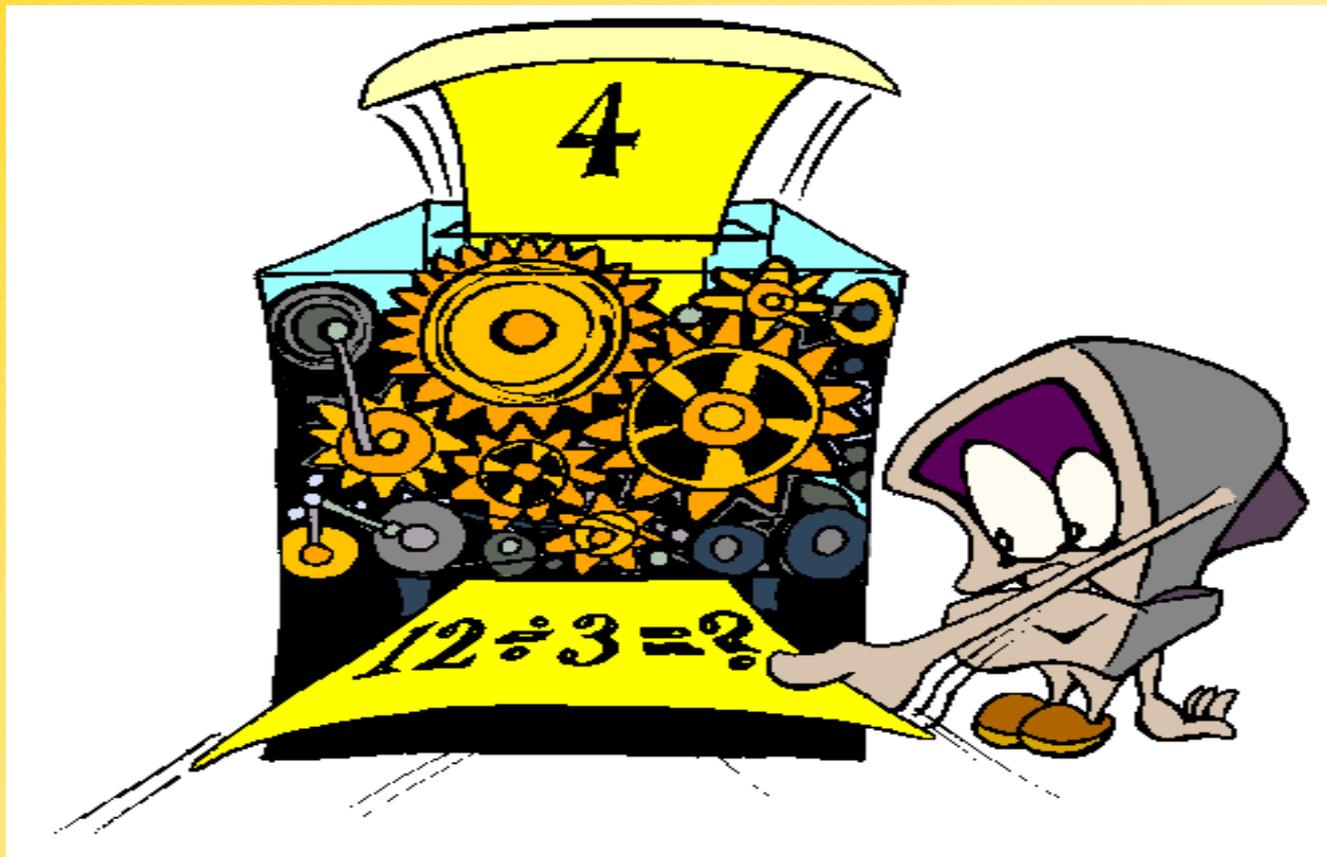
Clases

- Las definiciones de clases son **abstracciones**.
- Ellas definen el comportamiento de la abstracción.
- El *cómo* es logrado ese comportamiento no es materia de quien usa la clase, sino sólo de quien la implementa.
- Las clases son ***cajas negras***.
- En su implementación las clases definen también atributos para las abstracciones.

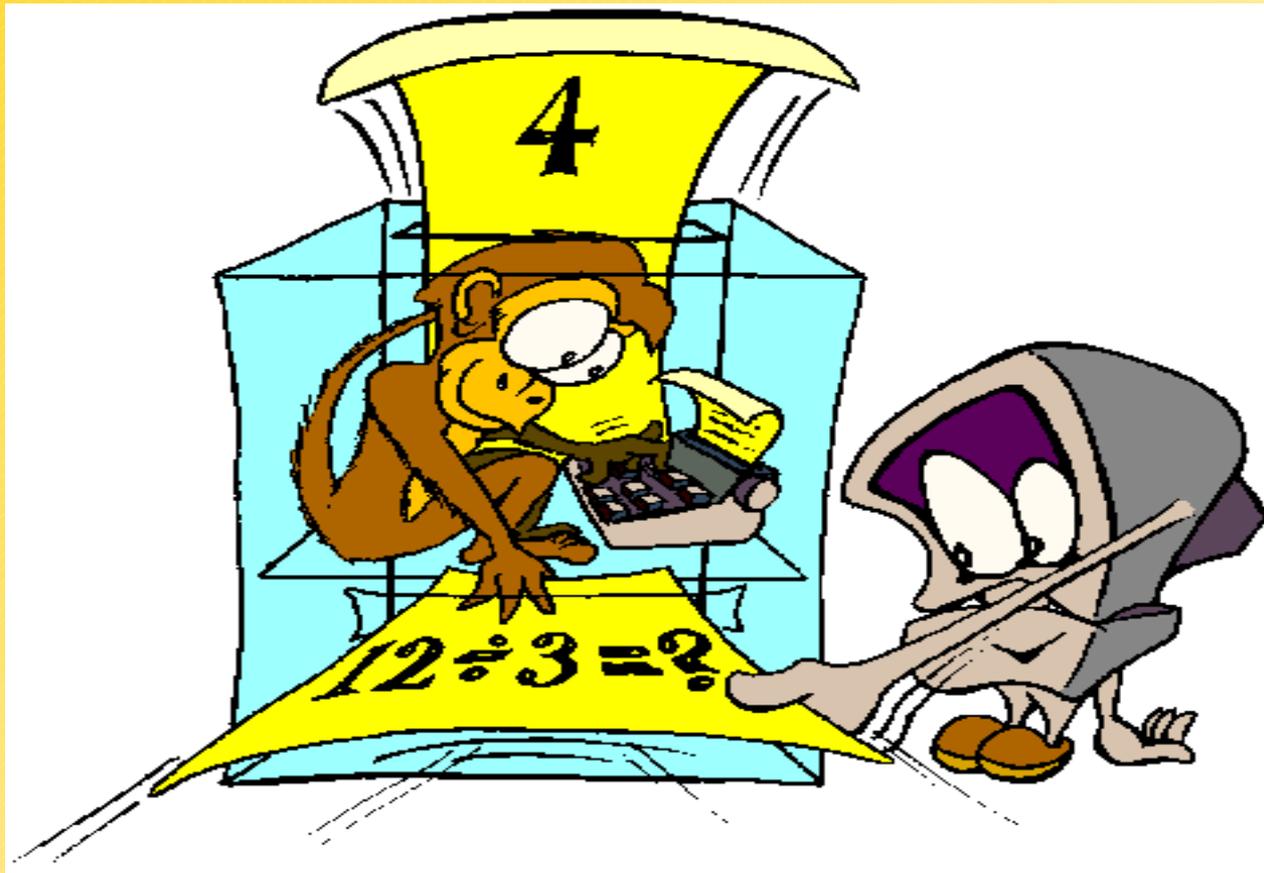
Calculadora



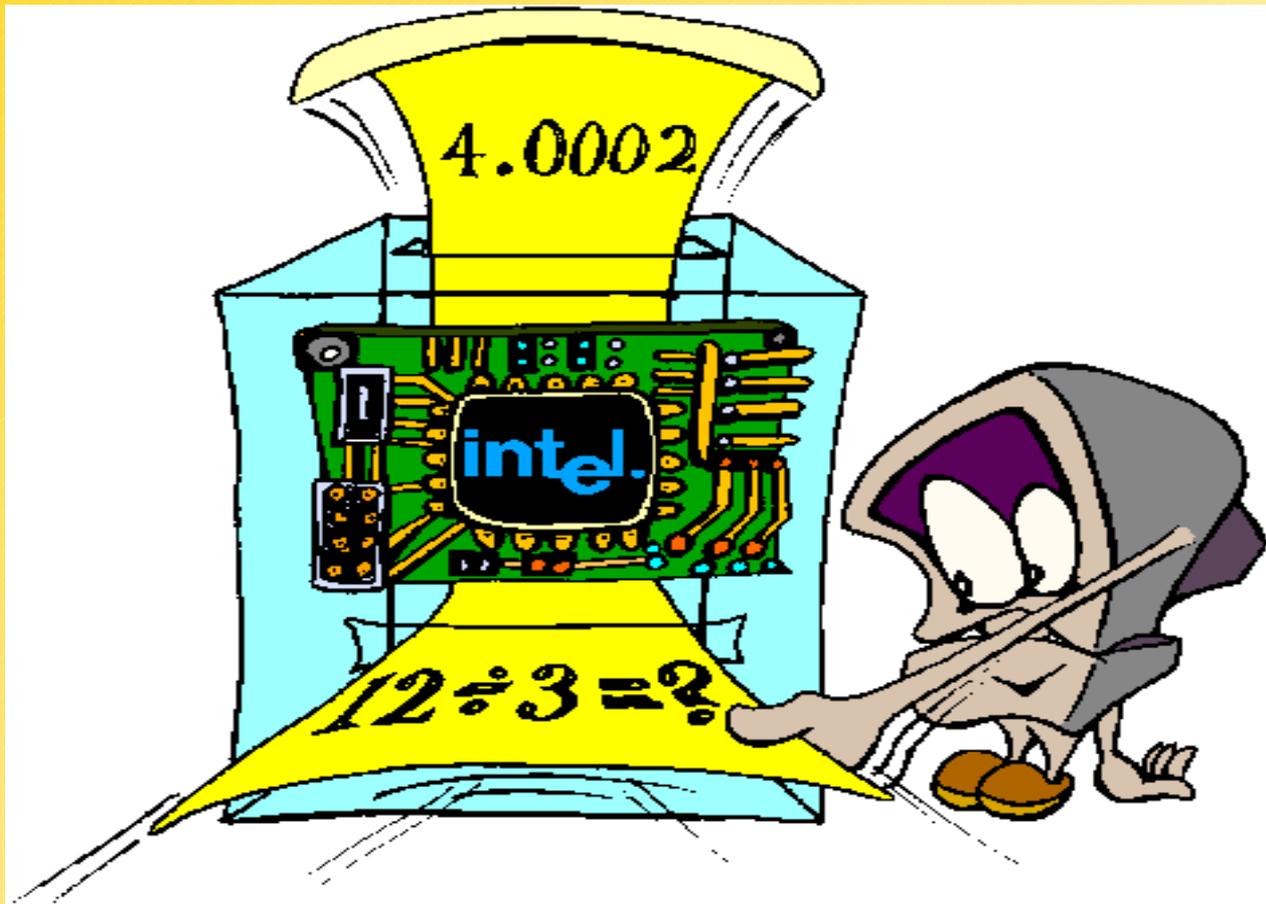
Calculadora

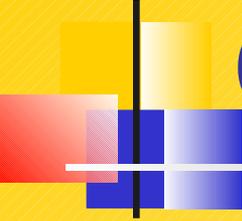


Calculadora



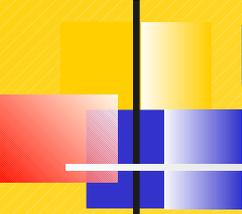
Calculadora





Clases

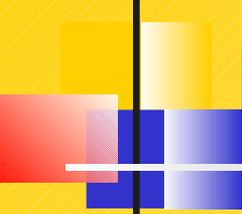
- Cada clase define **comportamientos** o responsabilidades o mensajes que pueden ser enviados a la clase
 - Puntos
 - Tienen distancia desde origen
 - puede ser trasladados, ...
 - Líneas
 - tienen largo, pendiente
 - puede interceptar otra, ...
 - Rectángulos tienen
 - largo, ancho, diagonal
 - perímetro, área,



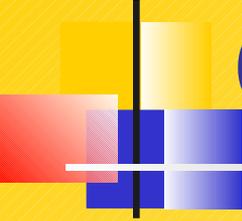
Una Clase- múltiple objetos

- Podemos **instanciar** (crear) múltiple objetos de una misma clase
 - crear puntos en diferente lugar del espacio
 - crear conjunto de líneas - todas con diferentes pendientes y largos

Clases e invocación de métodos

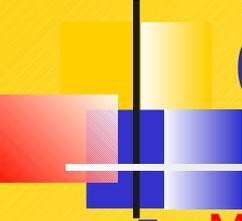


- Luego de crear un objeto, podemos aplicar **operaciones** de su clase a éste
 - Encontrar la distancia de un punto al origen
 - Mover un punto a una posición nueva
 - Determinar el largo de la línea
 - Preguntar si dos líneas se interceptan
 - *Formalmente*, decimos que **invocamos métodos** o **enviamos mensajes** de la clase a un objeto de la clase.



Clases

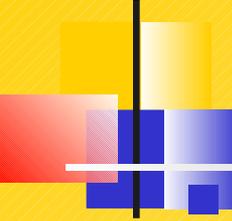
- Cada clase tiene dos **componentes**
 - **atributos**
 - especifican o califican el estado o las características individuales de un objeto
 - Punto: coordenadas x, y
 - Rectángulo: ancho, alto
 - RectanguloLleno: color (red, green, blue,)
 - **métodos**
- Sigue =>



Clases

Métodos

- Operaciones o servicios sobre objetos de una clase
 - Crear (constructor) y destruir objetos
 - obtener **valores** de los atributos de un objeto
 - Encontrar coordenadas x, y de un punto
 - Encontrar el largo de una línea
 - Encontrar el perímetro de un rectángulo
 - modificar los atributos de un objeto
 - trasladar un punto cambiando sus coordenadas
 - estirar un línea
 - expandir un rectángulo cambiando su ancho y alto



Ejemplo de clase

Rectangle

- Consideremos primero los métodos:

<code>Rectangle</code>	crea (construye) un rectángulo
<code>getWidth</code>	obtiene el ancho
<code>getHeight</code>	obtiene el alto
<code>setWidth</code>	cambia el ancho
<code>SetHeight</code>	cambia el alto

para hacer la clase más útil, definimos

<code>getPerimeter</code>	calcula el perímetro
<code>getArea</code>	calcula el área

Ejemplo de clase

Rectangle

- Consideremos primero los métodos:

Rectangle
getWidth
getHeight
setWidth
setHeight

Notar la convención de nombres en Java
operationTarget

cambia el ancho
cambia el alto

minúscula

Mayúscula inicial

para hacer la clase más útil, definimos

getPerimeter
getArea

No es obligación ..
**Fuertemente recomendada -
la API de Sun la usa**

Ejemplo de clase- Código java

- Rectangle.java

```
class Rectangle {  
    private double width, height;           // atributos  
  
    public Rectangle( double w, double h ) { // constructor  
        width = w;                          // fija atributos según  
        height = h;                          // parámetros  
    }  
  
    double Height() {                       // simplemente retorna  
        return height;                       // valor de atributo  
    }  
  
    double Width() {                        // simplemente retorna  
        return width;                        // valor de atributo  
    }  
  
    double getArea() {                     // retorna el valor de un atributo  
        return width*height;                // el cual es calculado  
    }  
  
    double getPerimeter() {                 // simplemente retorna  
        return 2.0*(width + height);        // el cual es calculado  
    }  
  
    void setHeight( double h ) {           // actualización (mutador)  
        height = h;                         // cambia el valor de un atributo  
    }  
  
    void setWidth( double w ) {            // simplemente retorna  
        width = w;                           // el cual es calculado  
    }  
}
```

Ejemplo de clase- Código java

```
class Rectangle {  
    private double width, height;           // atributos  
  
    public Rectangle( double w, double h ) { // constructor  
        width = w;                          // fija atributos según  
                                             // parametros  
  
        double Height() {                  // simplemente retorna  
            return height;  
        }  
    }  
}
```

■ Rectangle.java

Nombre de la clase

class Rectangle {

private double width, height; // atributos

.....

}

```
void setHeight( double h ) {           // actualización (mutador)  
    height = h;                         // cambia el valor de un atributo  
}
```

```
void setWidth( double w ) {  
    width = w;  
}
```

```
}
```

Ejemplo de clase- Código java

```
class Rectangle {  
    private double width, height;           // atributos  
  
    public Rectangle( double w, double h ) { // constructor  
        width = w;                          // fija atributos según  
                                             // parametros  
    }  
}
```

Nombre de la clase

```
double Height() {  
    return height;  
}
```

- Recta

class Rectangle

```
{
```

```
    private double width, height;           // atributos
```

```
.....  
}
```

Delimitadores de bloque

ización (mutador)
valor de un atributo

```
void setWidth( double w ) {  
    width = w;  
}  
}
```

Ejemplo de clase- Código java

```
class Rectangle {  
    private double width, height;           // atributos  
  
    public Rectangle( double w, double h ) { // constructor  
        width = w;                          // fija atributos según  
                                             // parametros  
    }  
  
    double Height() {                       // simplemente retornar  
        return height;                      // el atributo  
    }  
  
    // actualización (mutador)  
    // cambia el valor de un atributo  
    void setWidth( double w ) {  
        width = w;  
    }  
}
```

Nombre de la clase

class Rectangle

{

private double width, height;

// atributos

}

Delimitadores de bloque

Atributos

Ejemplo de clase- Código java

```
class Rectangle {  
    private double width, height;           // atributos  
  
    public Rectangle( double w, double h ) { // constructor  
        width = w;                          // fija atributos según  
                                             // parametros  
    }  
  
    double Height() {                       // simplemente retornar  
        return height;                      // atributo  
    }  
}
```

Nombre de la clase

class Rectangle {

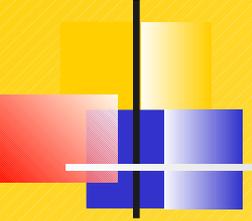
private double width, height; // atributos

Atributos

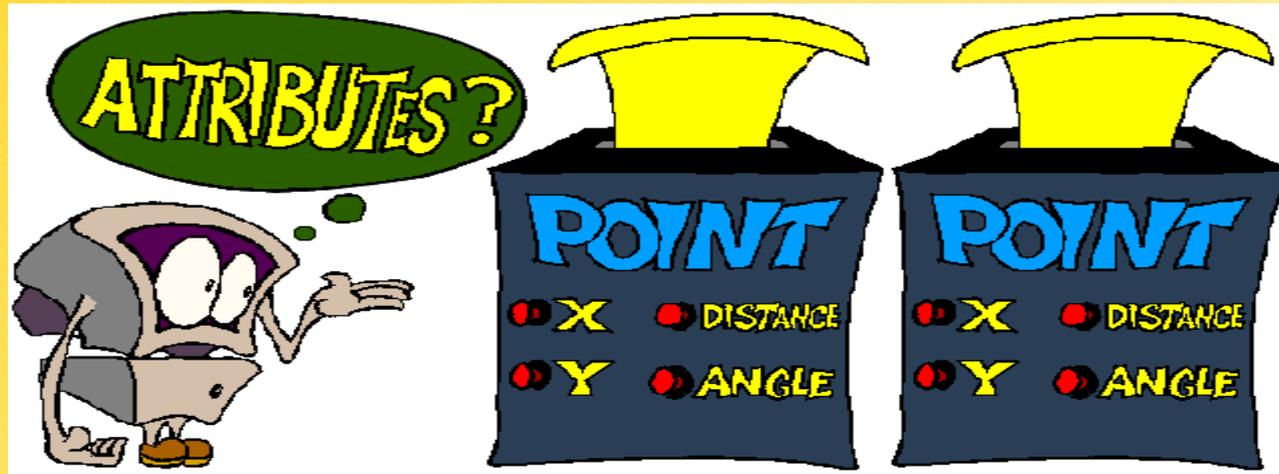
Notar: salvo excepciones, los atributos deben ser privados private!

En buenos diseños, las clases son *Cajas negras!*

¿Cajas negras?

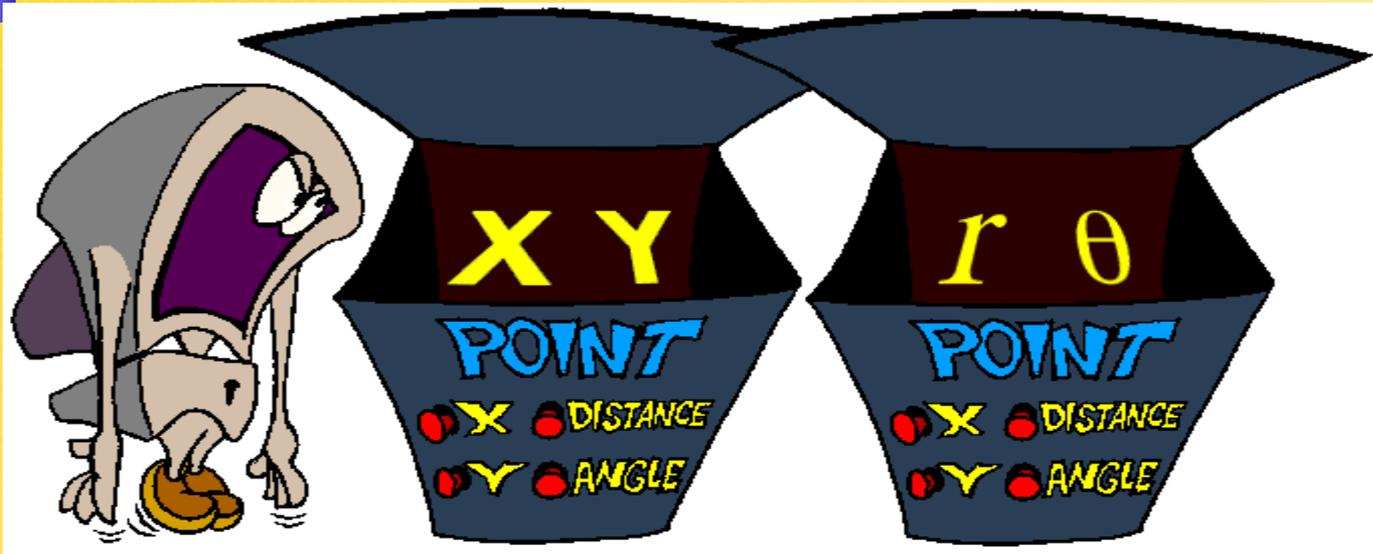
- 
-
- Una clase modela el comportamiento de algún conjunto de objetos similares en comportamiento.
 - Los métodos definen el comportamiento de una clase.
 - Atributos?
 - El implementador los elige
 - **No son** de incumbencia del usuario
 - Siempre y cuando la implementación sea correcta!
 - Ocultarlos en una "caja negra"
 - El acceso a ellos vía métodos
 - *Ejemplo Puntos en espacio 2-D*

Principio de ocultación de la información



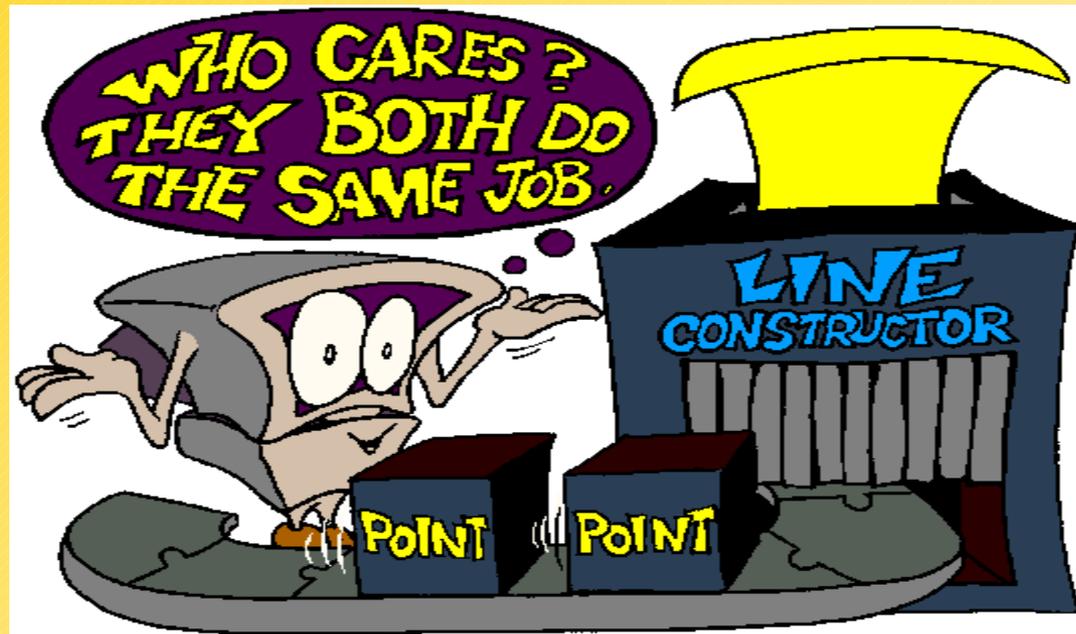
- Nuestro usuario intrigado puede ver
 - nombre de la clase
 - métodos:
 - XCoord, YCoord, Distance, Angle

Principio de ocultación de la información



- Mirando dentro, el usuario puede ver **dos** conjuntos de atributos diferentes!

Principio de ocultación de la información



- El usuario se da cuenta que *no necesita saberlo!*
- El usuario sólo quiere usar los puntos para hacer líneas!