Algoritmica II

Verano 2011

Pensando OO

Datos y código

- Todo programa está formado por 2 elementos: datos y código.
- Los datos es lo que se desea almacenar y procesar. Corresponden a la memoria.
- El código corresponde a las instrucciones que definen lo que se quiere hacer con esos datos. De esta manera definen el comportamiento.

Datos y código

- Todo programa debe organizar estos dos elementos.
- Existen 2 enfoques (o paradigmas) para hacerlo:
  - Programación basada en procesos.
  - Programación basada en objetos.

Programación basada en procesos

- El programa se organiza en base a las operaciones que se quieren realizar.
- Estas operaciones se organizan en grupos llamados procedimientos.
- Entonces el programa es un conjunto de procedimientos.

Programación basada en procesos

Problemas:

- Los datos quedan dispersos en el programa.
- Cualquier procedimiento puede hacer cualquier cosa con cualquier dato.
- En problemas muy grandes, este enfoque se vuelve caótico.

Programación basada en objetos

- El programa se organiza en base a los datos que se quieren almacenar y procesar.
- Estos datos se organizan en grupos, y para cada grupo se definen las operaciones que se quieren realizar con esos datos.
- Ese conjunto de datos y sus operaciones conforman un objeto.

Programación basada en objetos

- Entonces el programa es un conjunto de objetos que interactúan.

Entonces:

- Los datos quedan organizados en el programa.
- Una operación no puede hacer cualquier cosa con cualquier dato. Sólo puede hacerlo con los datos de su objeto.
- En problemas muy grandes, este enfoque permite reducir complejidad.

Clases y objetos

objetos

instancias de la clase

Pez

Clase PEZ
Los objetos de esta clase tienen **color**
y tienen la capacidad de **respirar bajo el agua**, **nadar** y **alimentarse**.

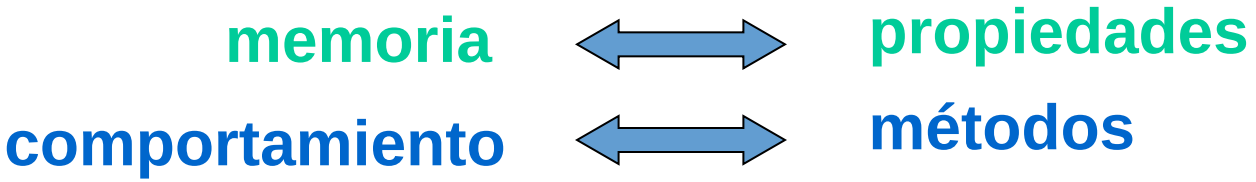
PECES
Estas son instancias de la clase pez.

Clases y objetos de software

memoria **comportamiento**

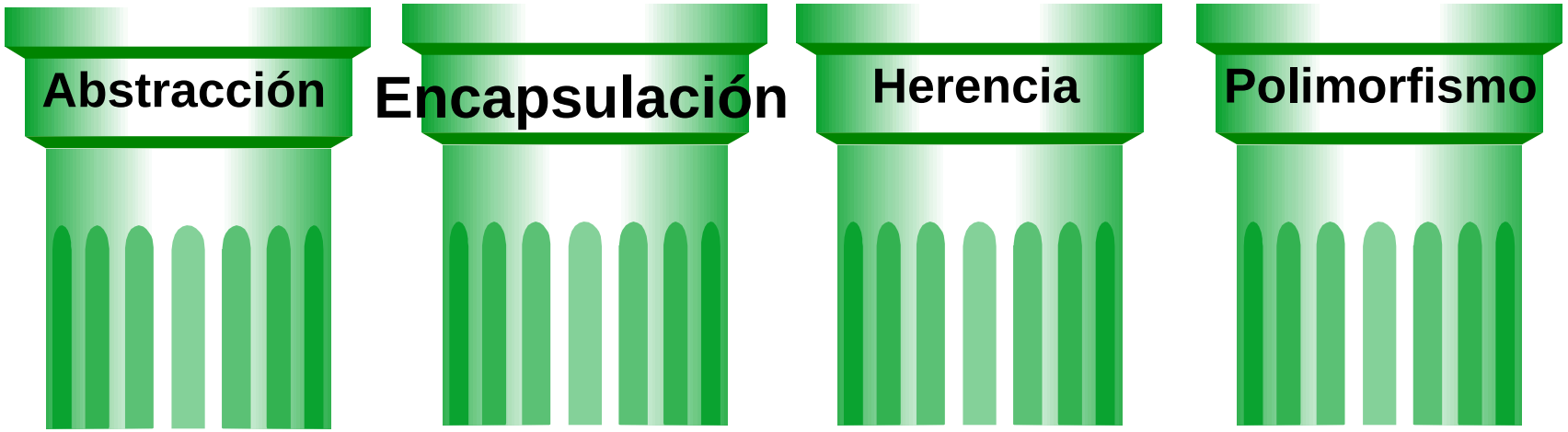
propiedades

métodos



Fundamentos de la P.O.O.

- La POO (Programación Orientada a Objetos) se basa en cuatro conceptos:



Abstracción

- Es el proceso de reconocer y seleccionar las características esenciales de los entes que forman parte de un sistema bajo estudio.
- Reduce la complejidad del problema real, permitiendo su manejo.
- Aplicada a la POO, la abstracción permite reconocer las propiedades de las clases de un software en desarrollo.

Abstracción (cont.)

Sistema de Ventas Casa Comercial

- Marca
- Modelo
- Nº Motor
- Nº chasis
- Color
- Equipamiento
- Valor de venta
- Fecha de venta
- Comprador
- Vendedor

Servicio Técnico:

- Propietario
- Marca
- Modelo
- Patente
- Fecha ingreso al taller
- Fecha salida de taller
- Kilometraje vehículo
- Tipo de falla

Registro Nacional de Vehiculos Motorizados:

- Propietario
- Marca
- Modelo
- Patente
- Nº Motor
- Nº chasis
- Color

Encapsulación

- Mecanismo que permite juntar el código y los datos que maneja en una sola estructura: la clase.
- Se forma como un envoltorio protector que mantiene al código y datos alejados de posibles interferencias o usos indebidos.

Encapsulación

- El acceso al código y a los datos se realiza de forma controlada a través de una interfaz bien definida.

Herencia

- Proceso en el cual un objeto adquiere las propiedades de otro.
- La herencia permite crear cada vez clases más especializadas (y complejas) a partir de las clases anteriores.
- Entonces, se forma una clasificación jerárquica de clases.

Polimorfismo

- “Muchas formas”, en griego.
- Permite que la misma interfaz se utilice para una clase general de acción. Entonces, el compilador selecciona la acción específica que se debe aplicar a cada situación.

Polimorfismo

- Con esto se reduce la complejidad.
- Se resume con la frase “Una interfaz, varios métodos”.

Posibilidades de la P.O.O.

- Usar clases previamente implementadas.

  Ejemplos:

  BufferedReader

  String

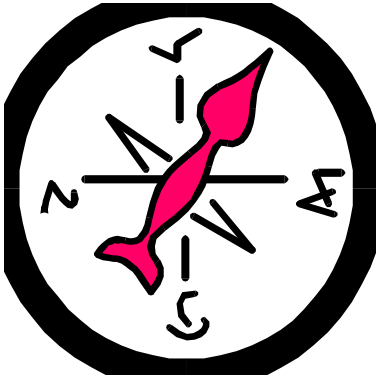
- Definir e implementar nuevas clases.

  Ejemplos:

  Persona

  Lista

  Auto



Orientación a Objetos

- OOP (Object Oriented Programming) es una idea distinta de otras en programación
- OOP es un paso en la evolución de previos abstracciones de programación

¿Por qué OOP es popular?

- La esperanza que rápidamente y fácilmente conducirá a aumentar la productividad y mejorar confiabilidad. De la mano viene Diseño Orientado a Objetos.
- El deseo de una transición simple de lenguajes existentes
- La similitud con técnicas de pensamiento sobre problemas en otras áreas

¿Por qué OOP es popular?

- La programación de un computador aún es una de las más difíciles tareas enfrentadas por el hombre; llegar a ser hábil en programación requiere talento, creatividad, inteligencia, lógica, la habilidad de construir y usar abstracciones, y experiencia.
- Programación Orientada al Objeto es una nueva forma de pensar sobre qué significa hacer cómputos, sobre cómo podemos estructurar información al interior de un computador.

Lenguaje y Pensamiento

- En otras palabras la forma como hablamos influye en la manera como vemos el mundo (y viceversa).
- Esto es válido no sólo para los lenguajes naturales (español, inglés, mapuche…) sino también para los lenguajes artificiales como los de programación (C, Pascal, C++, Java…)
- Ejemplo:
  - En MATLAB los loops son lentos de procesar, pero las matrices son muy rápidas. Se sugiere ver las soluciones operando matrices y no vía ciclos for.
- Corolario: los nombres de objetos e identificadores son relevantes. Después de un tiempo no hay pensamiento sólo nombres en el código.

Lenguaje y Pensamiento (Cont.)

- Hipótesis de Whorf: Trabajando en un lenguaje, es posible que un individuo imagine pensamientos o ideas que no pueden ser trasladadas o entendidas por individuos trabajando en otro contexto lingüístico.
  ¿Entendemos los problemas de origen étnico y religioso en algunos países del planeta?
  Blanco, blanco, blanco, blanco ¿Qué líquido toma la vaca?
  => nuestro conocimiento da forma a nuestras soluciones

- Conjetura de Church: Cualquier computación para la cual existe un procedimiento efectivo puede ser realizada por una máquina de Turing. (ésta dispone de una máquina de estados y una cinta donde se puede escribir y borrar). En los 60s se demostró que esta máquina podía ser emulada por cualquier lenguaje con la sentencia condicional.
  => Una máquina muy simple puede resolverlo todo

Lenguaje y Pensamiento (Cont.)

- Entonces en qué quedamos, estas dos ideas parecen contradictorias. Hay ideas que no son entendidas en otros contextos lingüísticos v/s la máquina de Turing con sólo sentencia if es capaz de hacerlo todo. ¿Para qué aprender otra cosa?

- Técnicas de orientación al objeto no proveen ninguna capacidad computacional nueva que permita resolver problemas no solubles por otros medios. Pero estas técnicas conducen a soluciones más fáciles y naturales (para el hombre) y favorecen la administración de grandes proyectos.

Computación como Simulación

- Ustedes pueden estar acostumbrados al modelo proceso-estado. Al estilo de la máquina de Von Newman. El computador sigue un patrón de instrucciones, organizadas en la memoria, saca valores desde varias localizaciones, los transforma, y pone resultados en otras localizaciones.
- Este modelo no ayuda mucho para entender cómo resolver problemas reales. No es la forma de pensar y ver las cosas.
- La visión de la OOP es crear un "universo" y es en muchas formas similar al estilo de la simulación llamado "simulación conducida por eventos"
- En OOP, tenemos la visión de computación como simulación.
- Cuando los programadores piensan en los problemas en términos de comportamientos y responsabilidades de objetos, ellos aprovechan su gran intuición, ideas, y entendimiento ganado con la experiencia diaria.
- Ver ejemplo líneas y puntos

Tratando con la complejidad

- Los problemas más complejos son comúnmente abordados por un equipo de programadores.
- La interconexión entre componentes es tradicionalmente complicada y por ello gran cantidad de información debe ser intercambiada entre los integrantes de un equipo.
- La incorporación de más gente puede **alargar** el proyecto en lugar de acortarlo.
- **El principal mecanismo para controlar la complejidad es la abstracción**. Ésta es la habilidad de encapsular y aislar localmente información de diseño.

Una nueva forma de ver el mundo

- **Supongamos que deseo enviar flores a mi abuelita. Una forma es ir a la florería y pedirselo a la vendedora. Le doy el tipo de flores y la dirección donde enviarlas.**
- **Yo busco un "*agente*" (la vendedora) y le paso un *mensaje* con el requerimiento. Es la *responsabilidad* de la vendedora el enviar las flores. Hay un *método*, conjunto de operaciones o algoritmo, usado por a vendedora para hacer esto. Yo no necesito conocer el detalle de este método.**
- **Las acciones son iniciadas en OOP por la transmisión de un mensaje (invocación de un método) a un agente (un *objeto*) responsable por la acción.**
- **Dos ideas: *Ocultar información* (hacer saber sólo lo indispensable o principio de "Information Hiding") y Reutilización. Sacarse la idea de tener que escribir todo y no usar servicios de otros (*delegar*).**
- **Dos importantes diferencias entre invocar Procedimientos y Mensajes**
  - **En mensajes hay un *receptor* designado, en procedimientos no.**
  - **La interpretación del mensaje (método) depende del receptor. Por ejemplo, mi esposa no actuaría igual si le pido enviar las flores.**
- **Usualmente el receptor de un mensaje no se conoce hasta tiempo de ejecución. Decimos que la *ligazón* entre mensajes (nombre de función, procedimiento o método) y el fragmento de código (implementación) usado para responder es determinada en tiempo de ejecución.**

Reutilización del software

- La gente se ha preguntado con frecuencia por qué el software no puede semejarse a la construcción de objetos materiales. Éstos normalmente son construidos a partir de otros elementos ya existentes y depurados.

- Por ejemplo: El acceso a una tabla indexada para buscar objetos es una operación común en programación; sin embargo, esta operación es re-escrita en cada nueva aplicación. Normalmente esto pasa porque los lenguajes tradicionales tienden a relacionar muy fuertemente el tipo del elemento con el código para insertar o buscar los elementos.

- El uso de técnicas de programación orientada al objeto debería conducir a generar gran número de componentes de software re-utilizables.

Ejemplo: Un Stack

- Este ejemplo ilustra las limitaciones de algunos lenguajes para ocultar información y desacoplar tareas.

```
int datastack[100];
int datatop = 0;
void init() {
   datatop=0;
}
void push (int val) {
   if (datato <100)
         datastack [datatop++] = val;
}
void top () {
   if (datatop >0 )
         return (datastack [datatop-1]);
}
int pop() {
   if (datatop >0)
         return (datastack [--datatop]);
   return 0;
}
```

• Los datos del stack no pueden ser locales a cada función
• Sólo hay dos opciones: Locales o Globales -> Globales
• Globales -> no hay forma de limitar la visibilidad de esos nombres.
• El nombre datastack debe estar en conocimiento de los otros programadores.
• Los nombre init, pus, top, pop, ya no pueden ser usados.

Ejemplo: Un Stack

- Definiendo el alcance dentro de un bloque (como en Pascal)
- begin
  var
    datastack: array [1..100] of integer;
    datatop: integer;
  procedure init; ….
  Procedure push(val: integer); ….
  Procedure pop : integer; ….
  …..
  end;
- Al dar acceso a la interfaz (o "protocolo" o nombre de funciones y profedimientos), también se está dando acceso a sus datos comunes (datatop) , al dar acceso los nombres quedan "tomados".
- La idea está contenida en los dos principios de David Parnas:
  1.- Proveer al usuario (otro programador, por ejemplo) toda la información necesaria para usar correctamente un módulo, y NADA MÁS.
  2.- Se debe proveer al implementador con toda la información que él necesita para completar el módulo, y NADA MÁS.

Responsabilidades

- Un concepto en OOP es describir comportamientos en términos de **responsabilidades**. Esto permite aumentar el nivel de abstracción y mejora la independencia entre agentes (importante para resolver problemas complejos).
- La colección de responsabilidades asociadas con un objeto es descrita por el término **protocolo**.
- No pregunte por lo que tú puedes hacer a tu estructura de datos, sino pregunta lo que tu estructura de datos puede hacer por ti.

Clases e Instancias

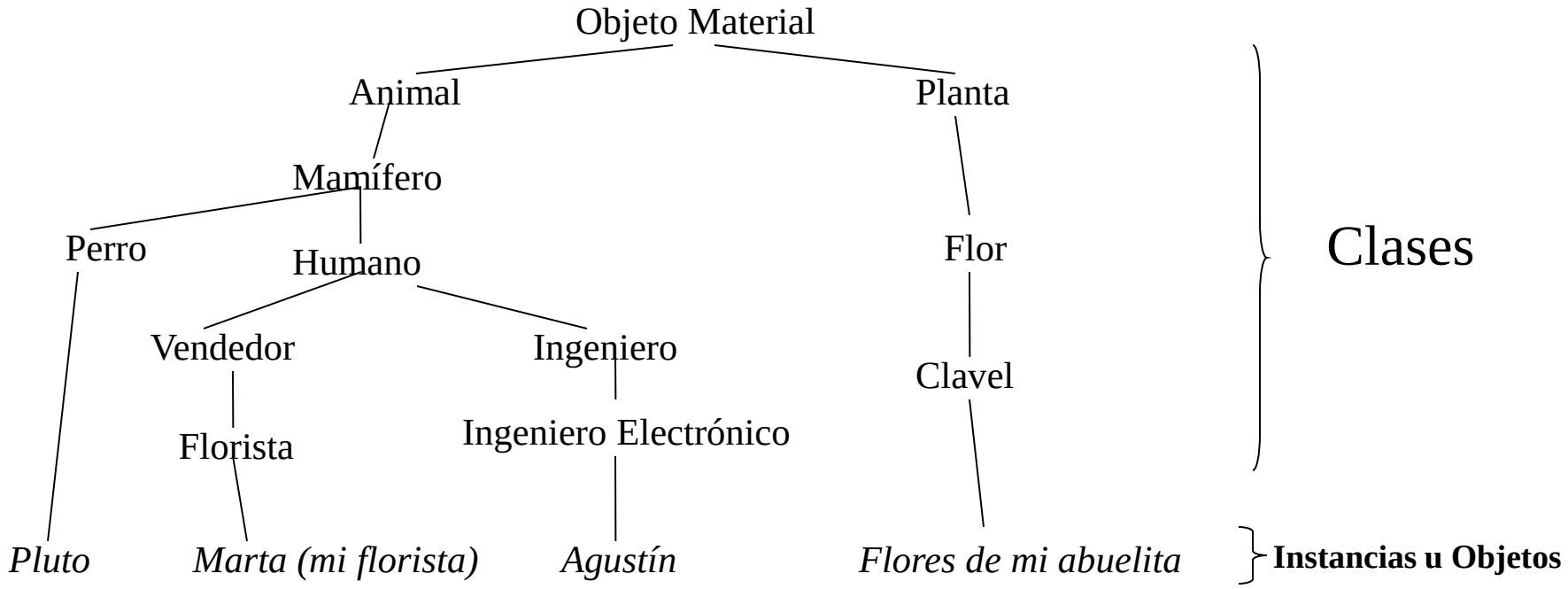
- Nosotros siempre tenemos una idea de las cosas más allá de ellas mismas. La vendedora es una instancia de una categoría o clase (por ejemplo Florista).
- Todos los objetos son instancias de alguna clase.
- Los objetos son entes que tienen nombre, comportamiento y estado.

Jerarquía de clases y herencia

- El hecho que **el conocimiento de una categoría más general es también aplicable a una categoría específica** se conoce como **Herencia**.
- Decimos que la clase Florista hereda los atributos de la clase Vendedor, y ésta hereda de la clase Humano, y ésta hereda de la clase Mamífero …. Se establece así una **Jerarquía de clases.**

Jerarquías de Clases

- Las clases pueden ser organizadas en estructuras de herencia jerárquicas.
- Una clase hijo (O subclase) hereda atributos de la clase padre. Una clase abstracta no posee instancias directas y es sólo usada para crear subclases. Por ejemplo Mamífero



Objetos-Mensajes, Herencia, y Polimorfismo

- Paso de mensajes: En OOP las acciones son iniciadas por un requerimiento a un objeto específico. (analogía: invocación de procedimiento es a procedimiento como mensaje es a método)

- Lo previo es nada más que un cambio de énfasis. ¿Qué es más natural: llamar a la rutina push con stack y dato como parámetros o pedirle a un stack hacer un push de un dato?
- Implícito está la idea que la interpretación del mensaje puede variar con diferentes objetos (polimorfismo). Los nombres no necesitan ser únicos. Se pueden usar formas más simples que conducen a programas más leíbles y entendibles.
- La Herencia permite a diferentes tipos de datos compartir el mismo código (=> menor tamaño de código y mayor funcionalidad).
- Polimorfismo: Hay varias formas de él. La idea básica es usar el mismo nombre o mensaje para referirse a cosas muy similares. ¿Por qué debería darle un nombre distinto a la función push cuando insertamos un real -float- o insertamos un carácter -char?

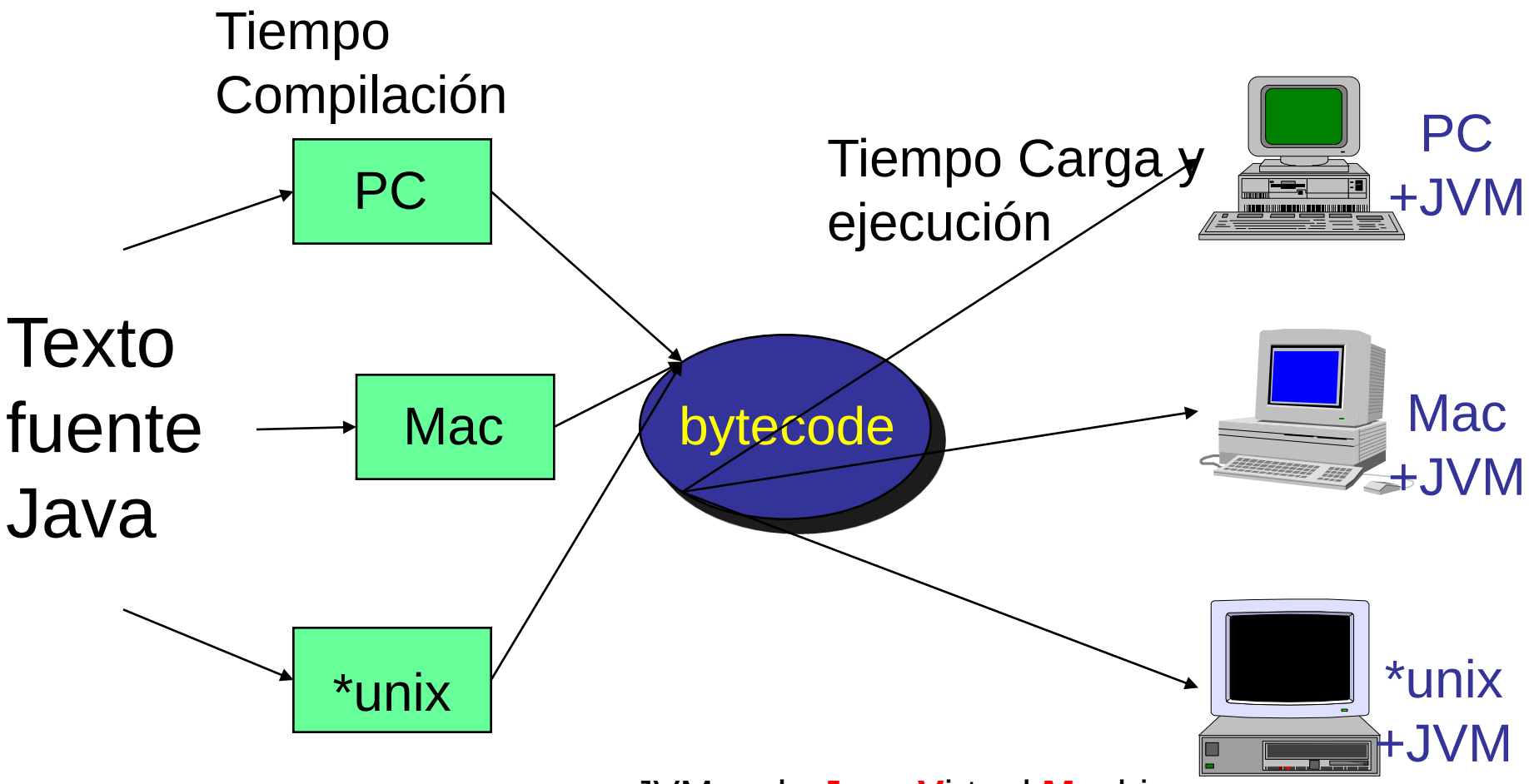
Introducción a Java y Diseño orientado a objetos

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…)

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

Compilación

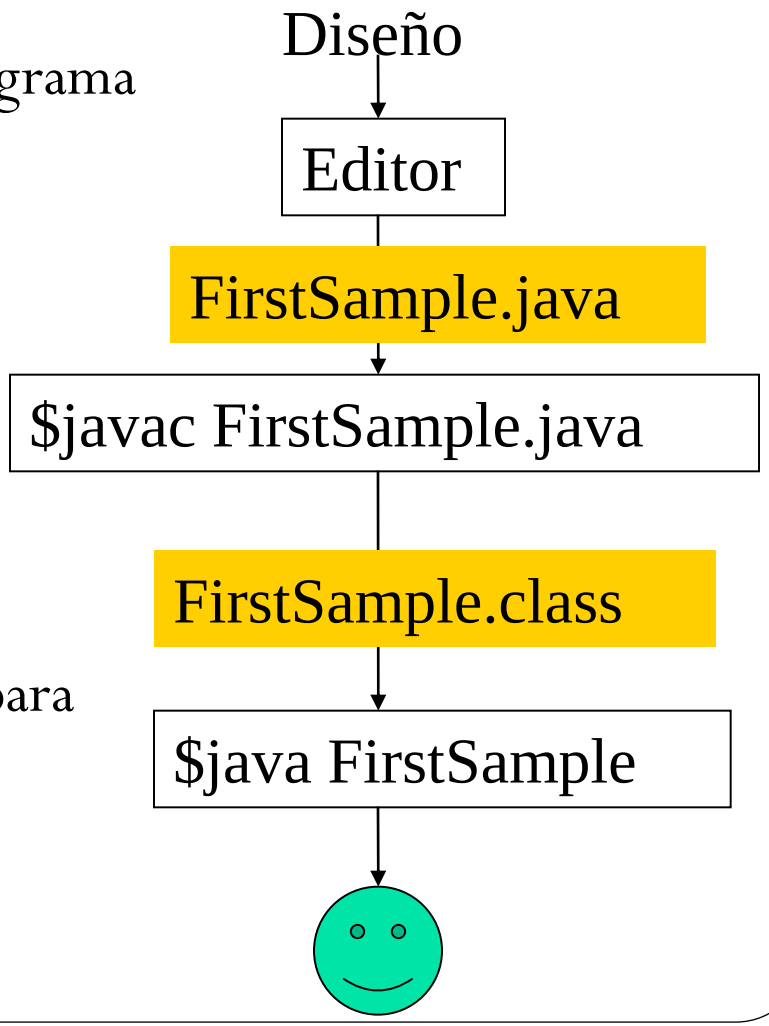


JVM es la **J**ava **V**irtual **M**achine,

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:
  - Jcreator
  - Eclipse
  - netbean
- Hoy otros, ver conveniencia.

Sistema de Desarrollo

- Lo puede bajar de SUN:
- http://java.sun.com
- Versiones:
  - Java EE (Enterprice 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	Clientes 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

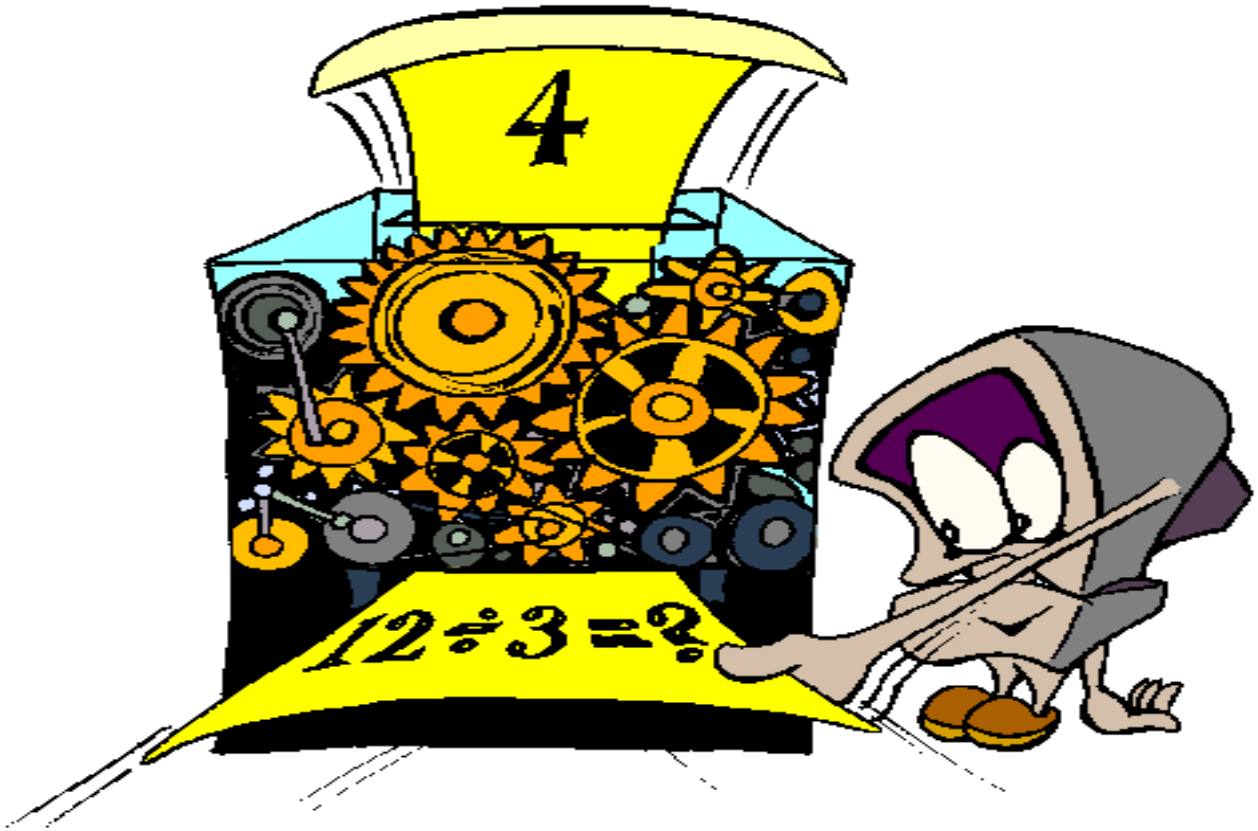
Clase	Comportamiento
Puente	• Colapsa con vientos sobre 50km/h. • Flexión de cubierta proporcional a la carga.
Dragon	• Puede ser creado con más de una vida. • Si le cae un rayo de más de 4 GVolts, se encoge y transforma en un montículo de polvo de oro.
Cuenta bancaria	• Cada cuenta puede tener distinta tasa de interés. • Sólo se permite retiros de hasta 200 K$ diarios.

Diseño orientado a objetos

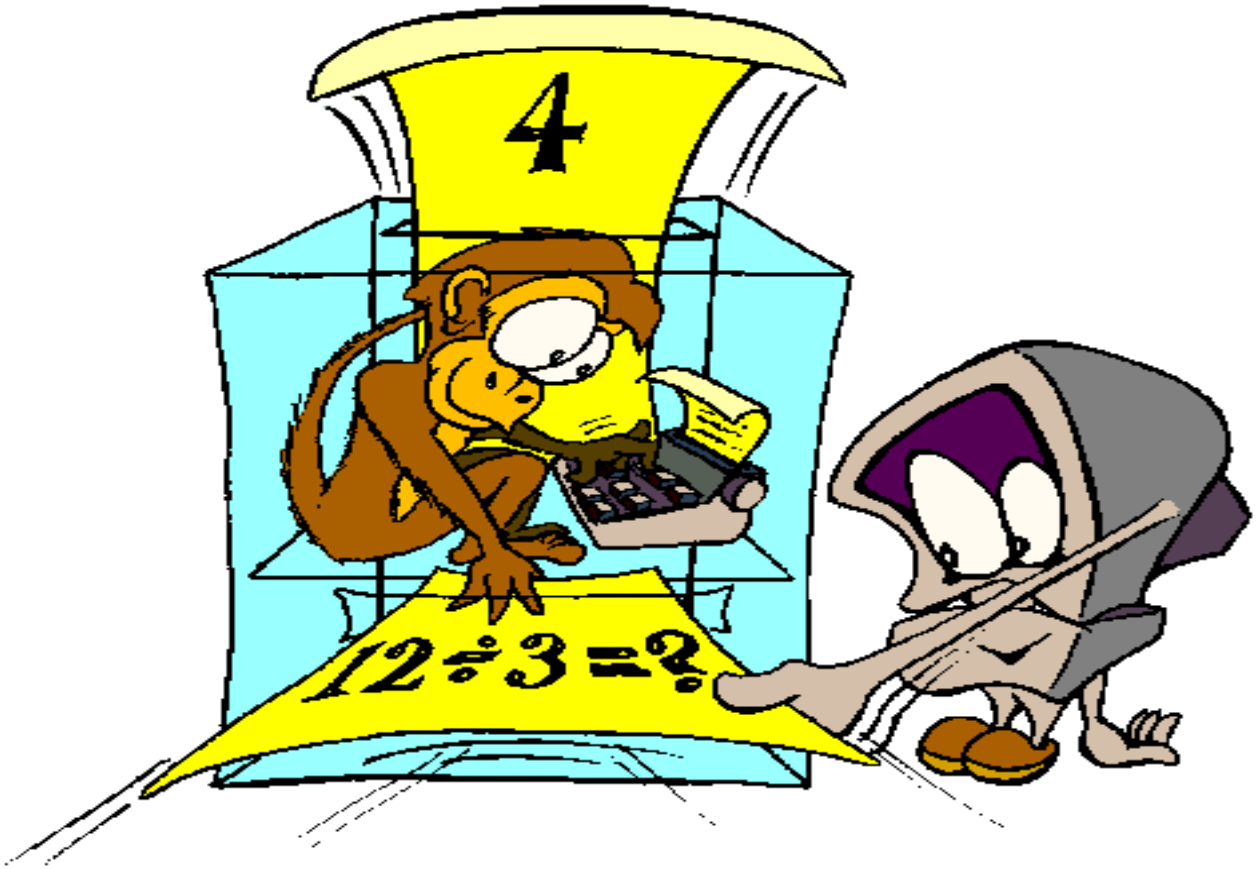
- Clases
  - Las definiciones de clases son abstracciones.
  - Ellas definen el comportamiento de la abstracción.
  - El *cómo* es logrado ese comportamiento no es materia de quien usa la clase, sino sólo de quien la implementa.
  - Las clases son *cajas negras.*
  - En su implementación las clases definen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

Calculadora

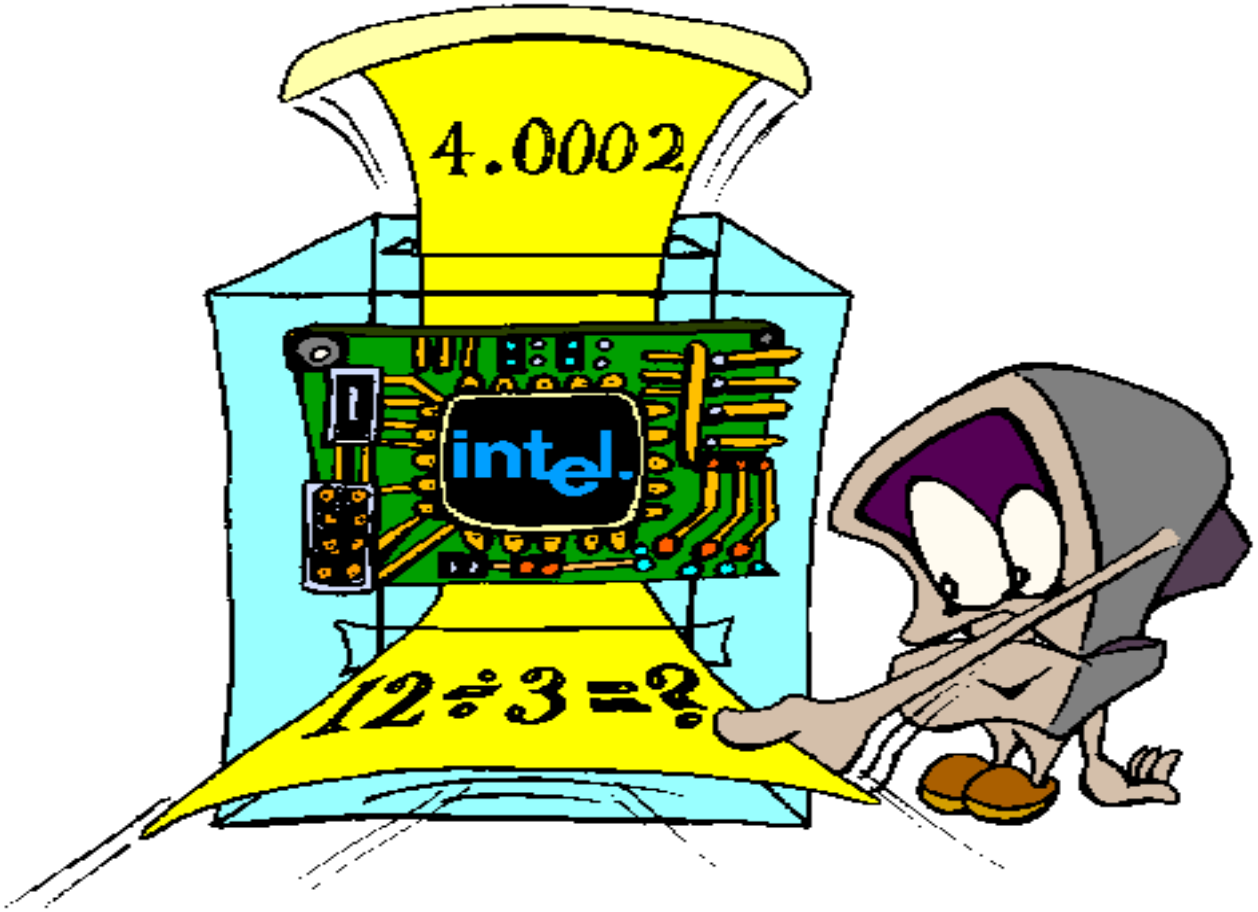
Calculadora



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.

Componentes de una clase en Java

Una clase en Java tiene:

- **atributos**: propiedades de los objetos de la clase.
- **métodos**: procedimientos que comparten los objetos de la clase.
- **constructores**: procedimientos que se ejecutan en el momento de la instanciación del objeto (tienen el mismo nombre 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:

`Rectangle`	crea (construye) un rectángulo
`getWidth`	obtiene el ancho
`getHeight`	obtiene el alto
`setWidth`	cambia el ancho
`SetHeight`	cambia el alto

para hacer la clase más útil, definimos

`getPerimeter`	calcula el perímetro
`getArea`	calcula el área

Ejemplo de clase

- `Rectangle`
  - Consideremos primero los métodos:

Notar la convención de nombres en Java
`operationTarget`

`Rectangle`
`getWidth`
`getHeight` obtiene el alto
`setWidth` cambia el ancho
`SetHeight` cambia el alto

para hacer la clase más útil, definimos

minúscula

Mayúscula inicial

`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;                             //   parámeteros
}

    double Height( ) {
return height;                           //  simplemente retorna
}                                        //   valor de atributo

    double Width( ) {
return width;
}

    double getArea( ) {
return width*height;                 //  retorna el valor de un atributo
}                                        //   el cual es calculado

    double getPerimeter( ) {
return 2.0*(width + height);
}

    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

- Rectangle.java

```
class Rectangle {
    private double width, height;                    // atributos

    public Rectangle( double w, double h ) {  // constructor
width = w;                              //   fija atributos según
                                        //   parameteros

    double Height( ) {
return height;                          //  simplemente retorna
}

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

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

Nombre de la clase

```
class Rectangle  {

    private double width, height;              // 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
                                    //   parameteros

    double Height( ) {
return height;                      //  simplemente retorna
```

Nombre de la clase

class Rectangle **{**

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

.....

}

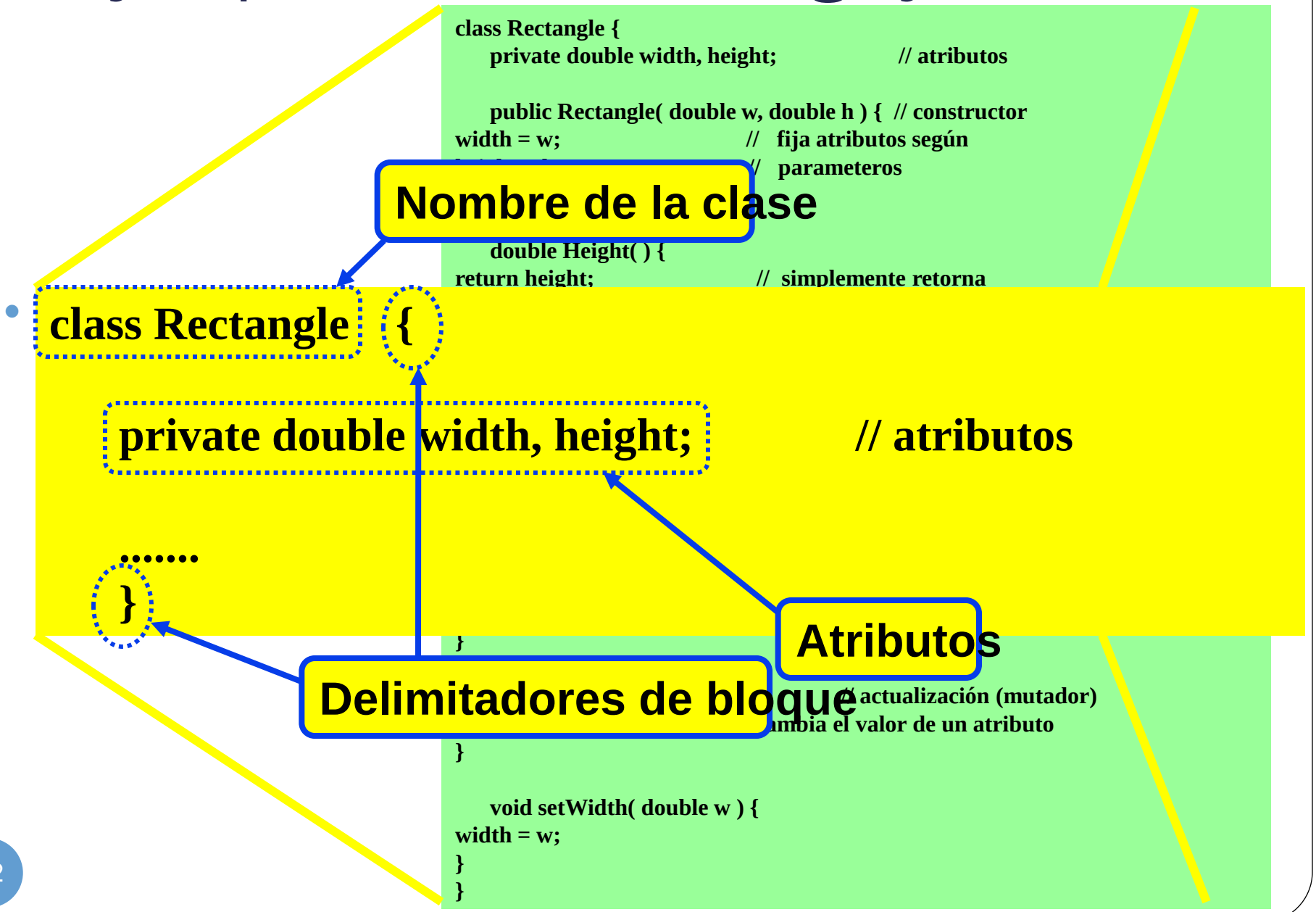
Delimitadores de bloque

```
}

                                    // actualización (mutador)
                                    // mbia el valor de un atributo
}

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

Ejemplo de clase- Código java



Ejemplo de clase- Código java

```
class Rectangle {
    private double width, height;                  // atributos

    public Rectangle( double w, double h ) {  // constructor
width = w;                              //   fija atributos según
                                        //   parameteros

    double Height( ) {
return height;                          //  simplemente retorna
```

Nombre de la clase

```
class Rectangle  {

    private double width, height;                  // atributos

    .......
    }
```

Atributos

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

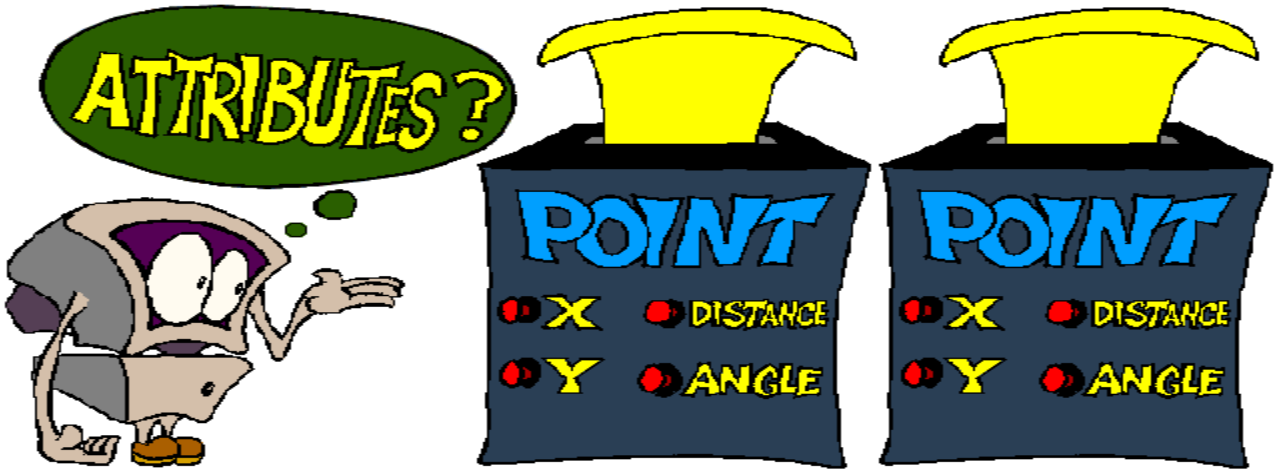
En buenos diseños, las clases son *Cajas negras!*

¿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

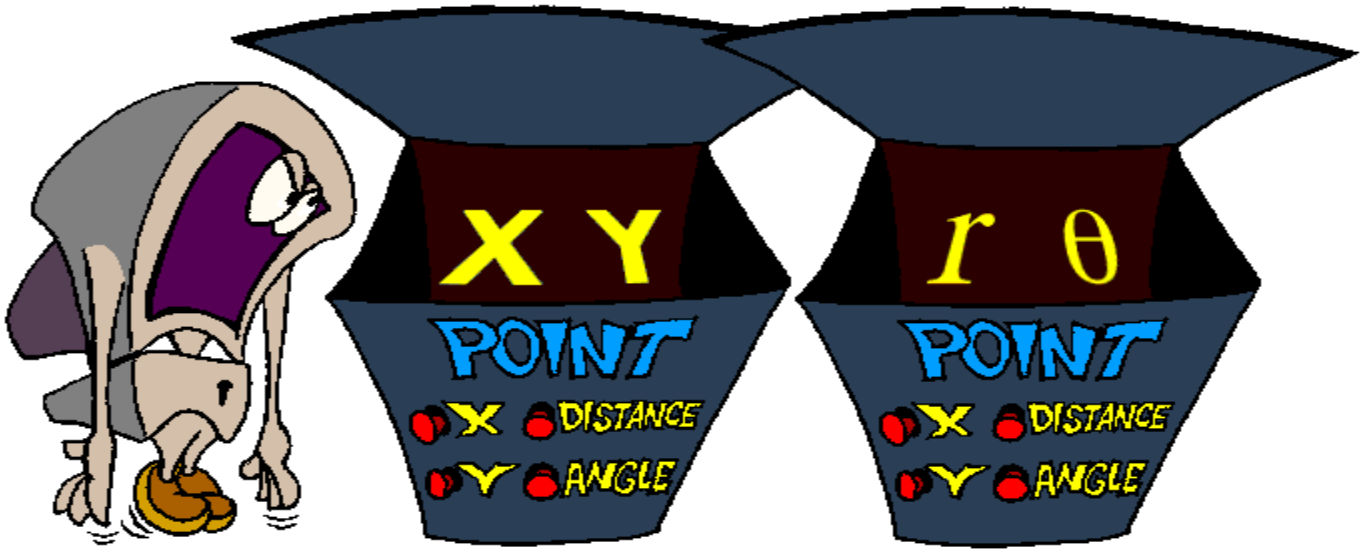
- *Ejemplo …. Puntos en espacio 2-D*

Principio de ocultación de la información



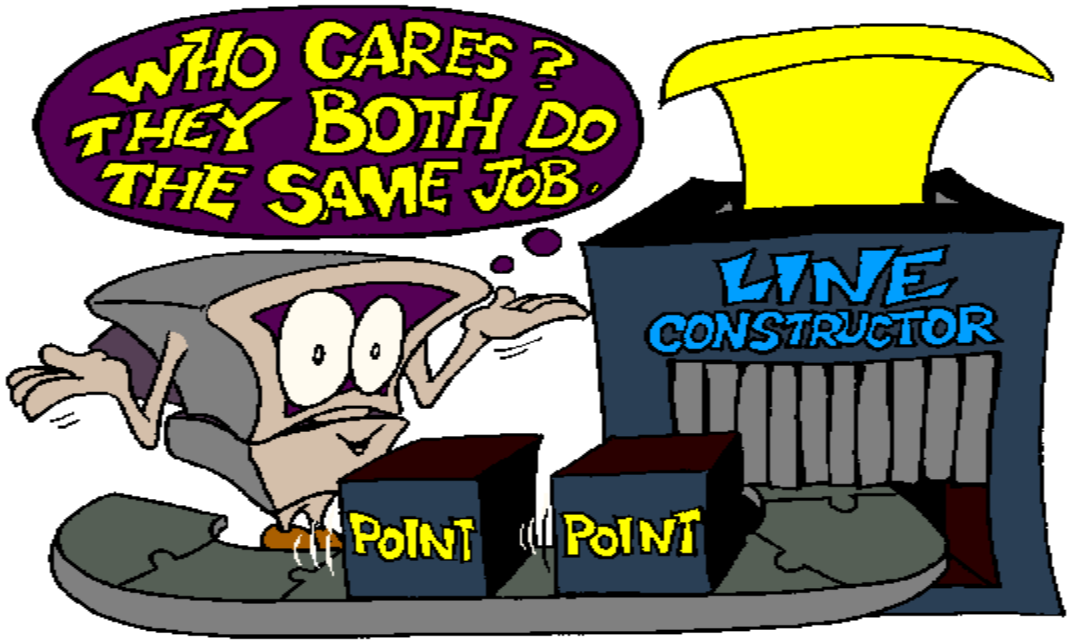
- Nuestro usuario intrigado puede ver
  - nombre de la clase
  - métodos:
    - `XCoord, YCoord, Distance, Angle`

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!

Estructuras fundamentales de la programación en Java

Primer programa en Java

- Todo programa debe tener al menos una clase.
- Toda aplicación Java debe tener el método main como el mostrado.
- System.out es un objeto al cual le invocamos el método println.

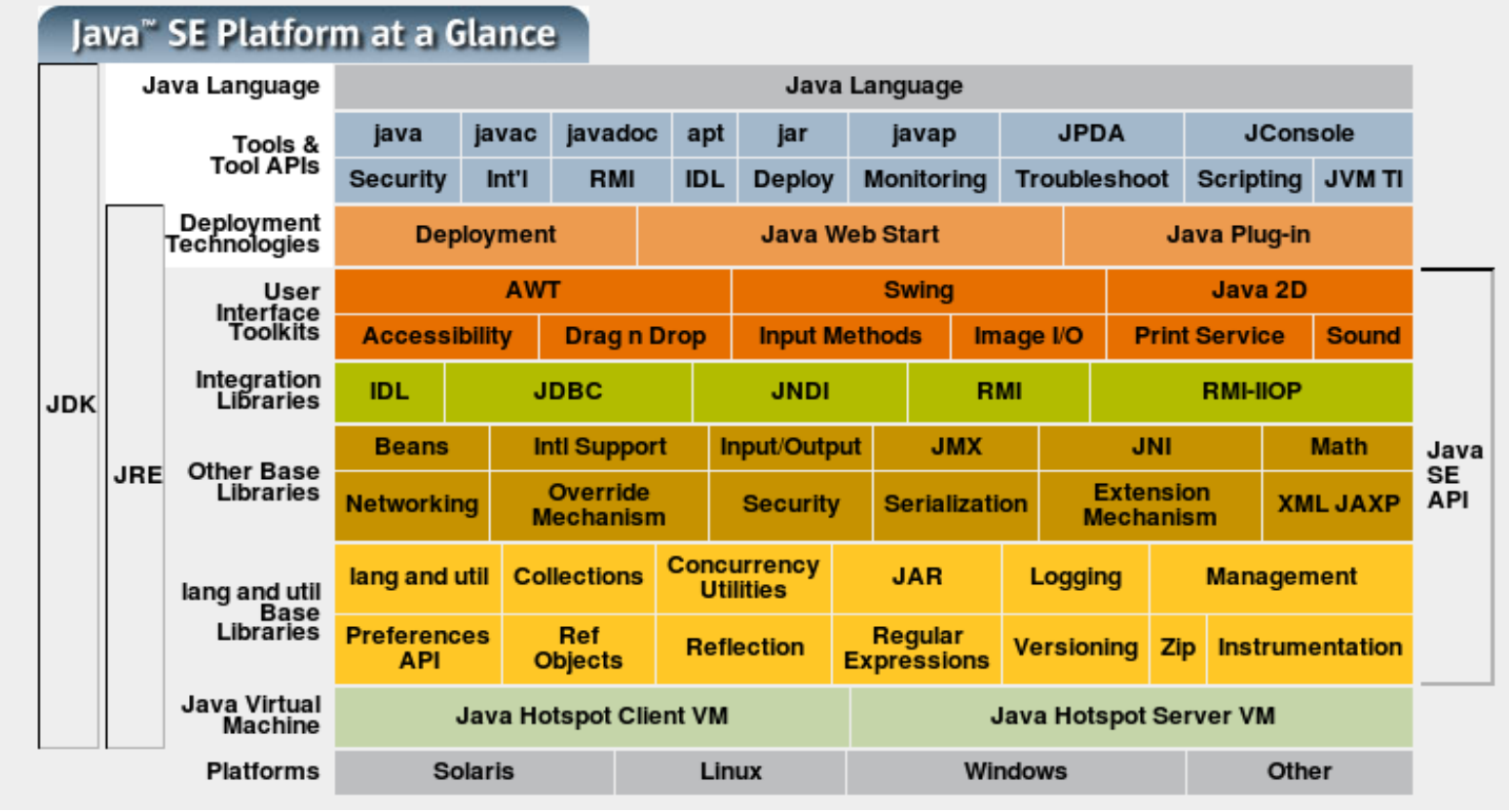
Nombre de archivo = FirstSample.java

```
public class FirstSample
{
  public static void main(String[ ] args)
  {
    System.out.println("We will not use 'Hello, Sansanos!'");
  }
}
```

- Ver: FirstSample.java

Trabajando con Java

- Desde http://java.sun.com/
  - Hay versiones para solaris, linux y windows.
- Ver: http://java.sun.com/javase/technologies/index.jsp



Instalación

- Hay otras versiones: Enterprice Edition (J2EE) y la Micro Edition (J2ME).
- Instalación en UNIX:
  - Incorporar el el path del compilador en el entorno al final de .bashrc o .bashrc_profile.
  - Por ejemplo: export PATH=/usr/local/jdk/bin:$PATH
- En Windows hacer lo equivalente (depende de su OS)
  - Control Panel -> System -> Environment. Avanzar hasta las variables de usuario y buscar la variable PATH. Agregar el directorio jdk\bin al comienzo. Ej c:\jdk\bin; otras rutas.

Ambientes de desarrollo

- Hay varios. Lo más básico es usar un editor de texto, escribir los programas, compilar y ejecutar en la línea de comandos. En esta opción yo uso emacs o xemacs como editor.
- Otros: kate en linux, netbean de Sun.
- Jedit: http://www.jedit.org/ También escrito en Java.
- Eclipse (usuarios señalan que requiere más máquina)
- Jcretor : http://www.jcreator.com
- Netbeens : http://www.netbeans.org

Aspectos básicos: Tipos primitivos (no son objetos)

- Booleano
  - boolean
    - true and false
- Enteros
  - int 4 bytes Ej: 24, 0xFA, 015
  - short 2 bytes
  - long 8 bytes Ej: 400L
  - byte 1 byte
- Punto flotante
  - float 4 bytes Ej: 3.14F (6-7 dígitos signif.)
  - double 8 bytes Ej: 3.14D (15 dígitos signif.)

Tipos primitivos (no son objetos)

- Carácter: `char`
  - Unicode
    - Usa *dos* bytes
    - Diseñado para internacionalización
    - Comillas simples: 'a', 'A', '!', '1', …
    - Forma hexadecimal '\u0008' (Unicode backspace)
    - El byte menos significativo corresponde al "ASCII" de 8 bits.
    - No visibles : Ej:
      - '\b' backspace '\t' tab
      - '\n' linefeed '\r' return
      - '\"' double quote '\'' single quote
      - '\\' el mismo backslash!

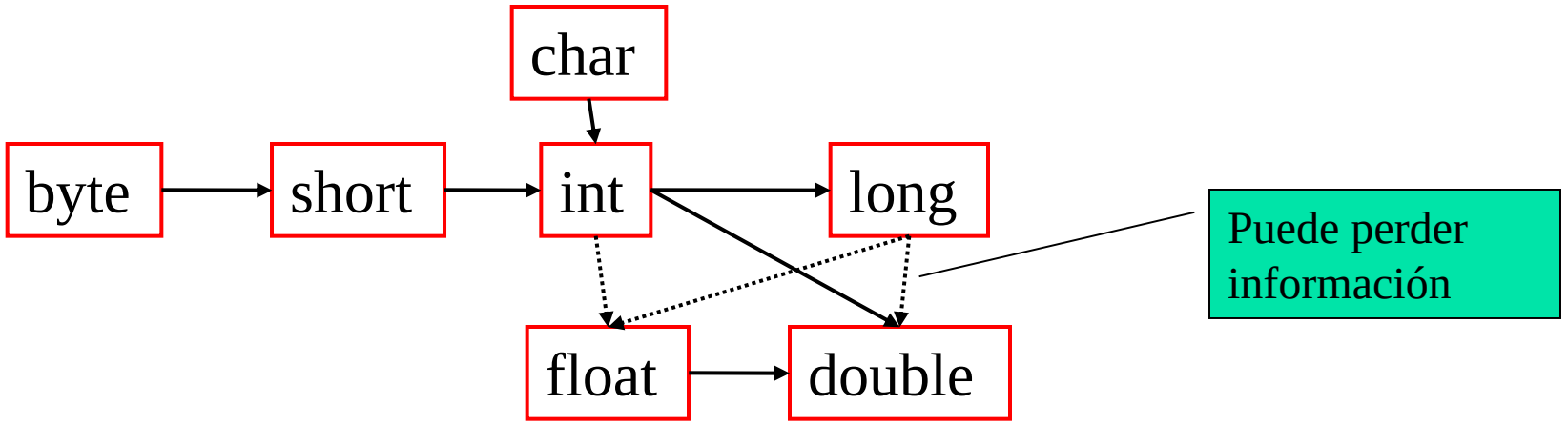
Constantes

- Se usa la palabra reservada *final*
- Ej: public final float CM_PER_INCH=2.54;
- Si deseamos crear sólo una instancia de esta constante para todos los objetos de una clase, usamos:

```
public class Constante
{
  public static final float MC_PER_INCH=2.54;
…}
```

  - El valor se accede: Constante.CM_PER_INCH

Cambios de tipo automáticos



Operadores y su precedencia

[] . ( ) ***(invocación)***	→
! ~ ++ -- + - (<tipo o clase>) new	←
* / %	→
+ -	→
<< >> >>>	→
< <= > >= instance of	→
== !=	→
&	→
^	→
|	→
&&	→
||	→
? :	←
= += -= *= /= %= &= |= ^= <<= >>= >>>=	←

String

- Java tiene una clase pre-definida llamada String.
- Todos los string son objetos y su comportamiento está dado por la clase .
- El operador + concatena strings. Si uno de los operandos no es string, Java lo convierte string y luego lo concatena.
  Ej: int nCanal=13;
  String estacion = “Canal”+nCanal;
- Para comparar dos strings, usar el método equals.
- El nombre de un objeto es una referencia al objeto (“dirección”), no el objeto mismo.

Entrada y Salida

- La salida de texto por consola es simple haciendo uso del objeto System.out. Es decir atributo out de la clase System.
- Hasta la versión 1.4 la entrada era bastante engorrosa. Esto se simplifica en V1.5
- Formas gráficas de entrada y salida se verán después.
- Las clases principales a estudiar son:
  - Java.io.PrintStream (desde Java 1.0), y
  - Java.util.Scanner (desde Java 1.5)

Salida de datos simple a consola

- Desde la versión 1.0 de Java existe la clase java.io.PrintStream.
- Define métodos para la salida de stream vía buffer.
- Los caracteres son puestos en memoria temporalmente antes de salir a consola.
- Los métodos son:
  - print(Object o): invoca método toString e imprime resultado.
  - print(String s): imprime string s.
  - print(*tipo_básico* b): imprime el valor de b
  - println(String s): Imprime s seguido de newline.

Entrada de datos simples por consola

- El objeto especial para efectuar entrada de datos es System.in; sin embargo, éste no ofrece métodos cómodos (es instancia de InputStream).
- Para facilitar la entrada de datos se creó a partir de la versión 1.5 la clase Scanner, en paquete java.util, la cual trabaja como envoltorio o recubriendo (wrapper) la clase InputStream.
- Scanner tiene varios métodos convenientes para la entrada de datos.
- Ver ejemplo: InputExample.java

Métodos de Java.util.Scanner

- Ver documentación
- Revisar métodos:
  - hasNext(): hay más datos en entrada?
  - next(): retorna próximo token.
  - hasNext*Type*(): Type es tipo básico. verdadro si hay dtal dato a continuación. Type es boolena, Byte, Double, Float, Int, Long y Short.
  - next*Type*(): retorna el dato del tipo *Type* a continuación.
  - Ver también: hasNextLine(), nextLine(); findInLine(String s);

Entrada de datos simple vía gráfica

- Otra forma de ingresar datos es vía la clase JoptionPane, en particular uno de sus métodos: JoptionPane.showInputDialog(promptString); este llamado retorna el string ingresado por el usuario.
- Ver ejemplo: InputTest.java

Sentencias (esto lo pueden estudiar por su cuenta)

- IF
- ```
  if( exp ) statement1;
  else statement2;
  ```

- ```
  if (a>b) x = a;
     else      x = b;
  else // es opcional
     if ( x[i] > max ) max = x[i];
  ```

Sentencias - Lazos

- while

```
while( exp ) statement1;
while( exp ) { statements; }
```

-
```
while (a>b) a = x[i++];
while ( x < 0 ) {
      x = getX( ... );
      y = y + x;
      }
```

- `while` permite evitar el viaje al bloque interno

Sentencias - Lazos

- do

```
do statement; while( exp );
do { statements; } while( exp );
```

-
```
do a = x[i++]; while( a>z );
```

-
```
do {
        x = getX( ... );
        y = y + x;
        } while ( x > 0 );
```

- do implica al menos un viaje

Sentencias - Lazos

- for

```
for( exp1; exp2; exp3 ) { s; }
```

- *equivalente a:*

```
exp1;
while ( exp2 )
        { s; exp3; }
```

- `for( k=0; k<n; k++ ) { s; }`

```
equivale a:
k=0;
while( k<n ) { s; k++; }
```

- *Patrón estándar para* `n` *iteraciones!*

Sentencias - switch

- ```
  switch( exp1 ) {
      case x1: s1; break;
      case x2: s2; break;
      default: s3;
      }
  ```
- Ejemplo:

```
  switch( x ) {
      case 1: y = a; break;
      case 2: y = b; break;
      default: y = c;
      }
```
```

Break y continue

- La sentencia break permite salir fuera del lazo de repetición sin terminarlo (además de su uso en switch).
- También puede ser usada en conjunto con un rótulo para salir fuera de cualquier bloque. El rótulo va inmediatamente antes del bloque en cuestión.
- La sentencia continue transfiere el control de flujo al encabezado del lazo más interno.

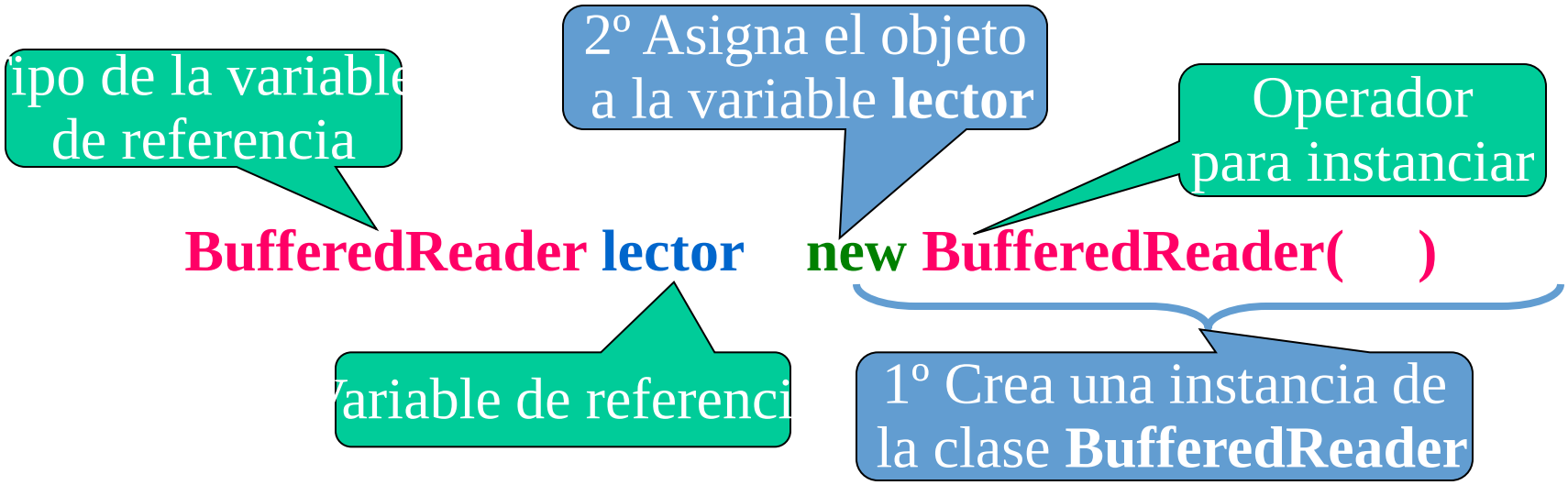
Clases para tipos de datos primitivos

- Envoltorios (Wrappers)
  - Crean objetos para los tipos estándares.
  - `java.lang`
    - `Boolean`
    - `Integer`
    - `Long`
    - `Character`
    - `Float`
    - `Double`
  - Un método importante en estas clases nos permite transformar un string que contiene números en un tipo básico. Ej: int ***a*** = Integer.parseInt("3425");
    hace que ***a*** tome el valor 3425.
    Se usó en ejemplo InputTest.java

Objetos y Clases en Java

Uso de clases y objetos

- Los objetos deben instanciarse
- Cada objeto tiene su propia identidad
- Cada objeto se "referencia" desde una variable



Instanciación de objetos

- Para utilizar un objeto primero se debe definir una variable que lo irá a referenciar, con el formato:

```
Clase variable
```

- Luego se debe crear el objeto (instancia de clase), de la siguiente forma:

```
variable = new Constructor(lista de parámetros)
```

Creación de objetos nuevos

- Se usa el constructor de la clase
  MiClase a = new MiClase();
- Todos los objetos son creados en el heap (memoria asignada dinámicamente durante la ejecución).
- Lo que se retorna es una referencia al nuevo objeto (puede ser pensada como puntero).
- Nota — no existe destructor (en C++ sí)
  Java tiene un proceso de recolección de basura (Garbage Collection) que automáticamente recupera zonas no referenciadas.

Constructores

- Tiene igual nombre que la clase
- Pueden tener parámetros
- Son invocados principalmente con **new**
- No tiene tipo retornado
- No **return** explícito
- Java provee constructor por defecto **()**
- Podemos proveer uno o más constructores. Esto es un tipo de sobrecarga de métodos (igual nombre con distintos parámetros)
- El compilador busca el constructor usando firma nombre constructor + lista de parámetros

Constructores

- Inicializa objetos nuevos:
  - 1. Localiza memoria
  - 2. Asigna valores por defecto a variables (0, 0.0, null, …)
  - 3. Llama constructor de Superclase (más adelante)
  - 4. Sentencias restantes son ejecutadas
- La primera sentencia puede ser:
  - super(…) para llamar al constructor de la clase base (o padre o superclase)
  - this(…) invoca a otro constructor

Referencias

- Los objetos son referenciados
- Esta es una forma “controlada” de usar: Direcciones y punteros
- Al declarar una variable de una clase obtenemos una referencia a la variable.
- En caso de tipos primitivos (8) se tiene la variable y acceso directo (no es referencia)
  - byte, short, int, long, float, double, char, boolean

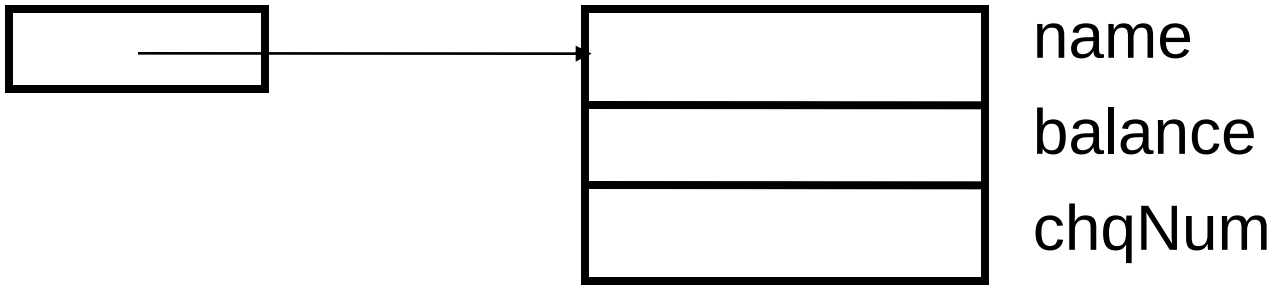
Definiendo variables

Cheque pejAcct;

pejAcct  Referencia nula

pejAcct.deposit(1000000); // error

pejAcct = new Cheque("Peter", 1000, 40);

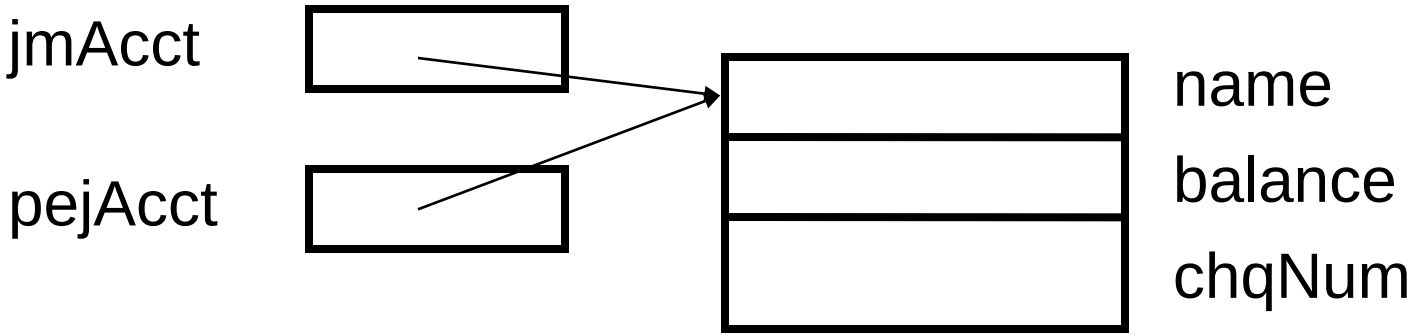


Este ejemplo asume que la clase Cheque ya existe y posee miembros datos: name, balance y chqNum

Asignación

Cheque jmAcct;

jmAcct = pejAcct;

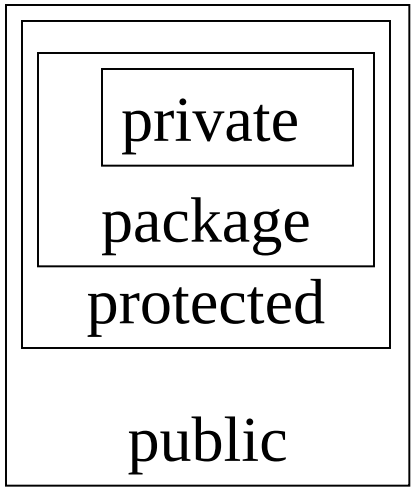


Implicancias de referencias

- La identidad de objetos son **referencias**
  - referencia significa puntero (i.e. no el contenido)
- = es copiar la referencia
  - Usar método **clone** para crear copia del objeto completo.
- == es comparación de referencias
  - Usar **equals** para comparar contenidos
- aMethod(pejAcct) pasa un referencia
- aMethod(tipo_básico) pasa el valor
- return pejAcct retorna una referencia
  - Usar clone para crear una copia, y luego retornarla

Control de acceso

Modificador de acceso	*Visibilidad*
public	Todas partes
protected	en sub-classes & pkg
"omitido"	En el paquete
private	Sólo en la clase



Documentación

- Para la clase ponerla inmediatamente antes de la clase y ser encerrado entre /** y */
- Para los métodos: usar los rótulos
  - @param *variable* descripción
  - @return descripción
  - @throws descripción de clase
- Para los datos públicos: /** …*/
- Comentarios Generales:
  - @author nombre
  - @version texto
  - @since texto
  - @see link
  - Ejemplo: @see cl.utfsm.elo.Employee#raiseSalary(double)

Documentación

- Se pueden usar todo tipo de rótulos html incrustados.
- ¿Cómo generar la documentación?:
  - javadoc -d docDirectory *.java
- Para la documentación de un paquete:
  - javadoc -d docDirectory nameOfPackage
- Ejemplo:
  - Account.java
  - index.html generado con javadoc -d AccountDoc *.java

Rutas para clases

- Primero incluir la ruta del compilador y máquina virtual java en la variable PATH.
- Luego la ruta para la búsqueda de todas las clases: CLASSPATH
  - El compilador y el interprete java buscan los archivos en el directorio actual.
  - Si el proyecto está compuesto por varias clases en diferentes directorios, javac y java buscan las clases en los directorios indicados en la variable de ambiente CLASSPATH.
- En Linux ELO ésta se configura con
  - export CLASSPATH=/home/user/classdir1: /home/user/classdir2:.
- El Windows también se debe fijar la variable de ambiente.

Instanciación de objetos: ejemplo

CajaAhorro

- Tiene los siguientes métodos:
  - **depositar(int monto)** : permite abonar el valor de monto a la cuenta.
  - **girar(int monto)**: permite registrar un giro por el valor de monto.
  - **obtenerSaldo()**: retorna el saldo de la cuenta (valor int).
  - **obtenerTransacciones()**: retorna la cantidad total de transacciones (giros y depósitos) que se han hecho sobre la cuenta (valor int).
- Y el siguiente constructor:
  - **CajaAhorro()** : inicializa la cuenta con saldo y contador de transacciones en cero.

Instanciación de objetos: ejemplo (cont.)

```
public class Ejemplo {
  public static void main (String arg[]) {
     CajaAhorro cta1 = new CajaAhorro();
     cta1.abonar(1000);
     cta1.abonar(500);
     cta1.girar(300);
     int saldo = cta1.obtenerSaldo();
     int trans = cta1.obtenerTransacciones();
     System.out.println( "El saldo es" + saldo );
     System.out.println( "Se han hecho" + trans + "transacciones" );
  }
}
```

Instanciación de objetos: ejemplo (cont.)

```
public class Ejemplo {
  public static void main (String arg[]) {
     CajaAhorro cta1 = new CajaAhorro();
     CajaAhorro cta2 = new CajaAhorro();
     cta1.abonar(1000);
     cta2.abonar(500);
     cta1.girar(800);
     System.out.println( "El saldo de cuenta 2 es" +
                         cta2.obtenerSaldo() );
     System.out.println( "Hubo" + cta1.obtenerTransacciones() +
                         "transacciones en cuenta 1" );
  }
}
```

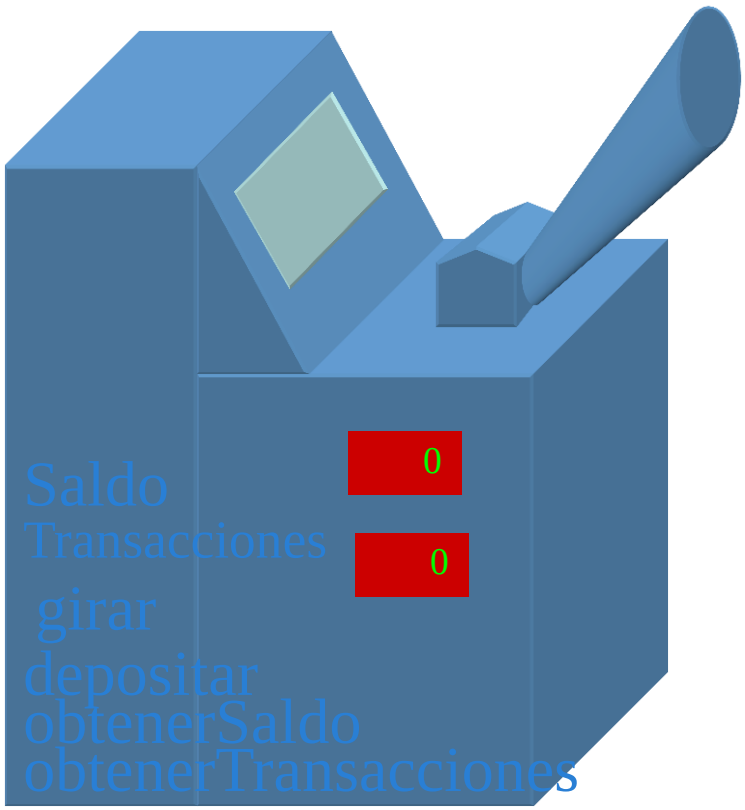
→ *500*

→ *2*

```
import java.io.*;
class Banco {
   public static void main(String argum[]) throws IOException {
      BufferedReader op = new BufferedReader(new InputStreamReader(System.in));
      String opc;
      int cant,opcion;
      CajaAhorro cuenta = new CajaAhorro();
      boolean continuar = true;
    System.out.println("Seleccione una opción");
    System.out.println("1.-Abonar  2.-Girar  3.-Cons. Saldo  4.- Salir");
    opcion= Integer.parseInt( op.readLine() );
          switch (opcion) {
            case 1:
               System.out.println("Ingrese la Cantidad a abonar");
               cant= Integer.parseInt( op.readLine() );
               cuenta.abonar(cant);
               break;
            case 2:
               System.out.println("Ingrese la Cantidad a girar");
               cant=Integer.parseInt( op.readLine() );
               cuenta.girar(cant);
               break;
            case 3:
               System.out.println("Saldo :" + cuenta.obtenerSaldo());
               System.out.println("Trans.:" + cuenta.obtenerTransacciones());
               break;
            case 4:
               continuar = false;
         }
      }
   }
}  // fin clase
```

Ejemplo de programa que utiliza una clase

Ejemplo: clase CajaAhorro

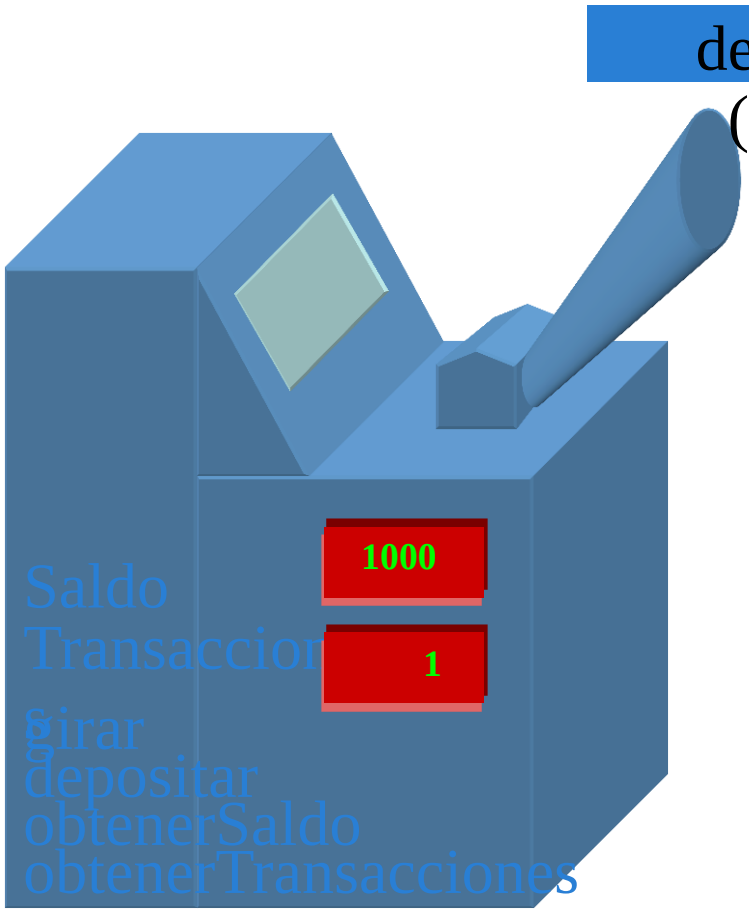


Objeto

Aplicación

Se requiere la instanciación de un objeto de la clase CajaAhorro.

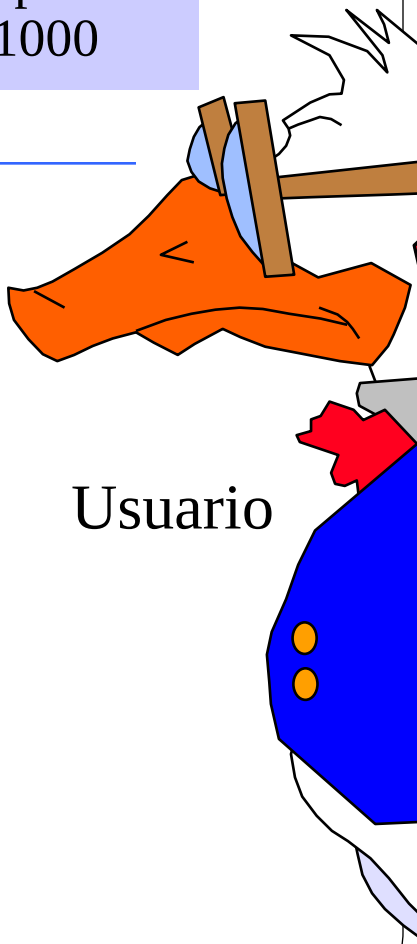
Ejemplo: depositar



Objeto

depositar
(1000)

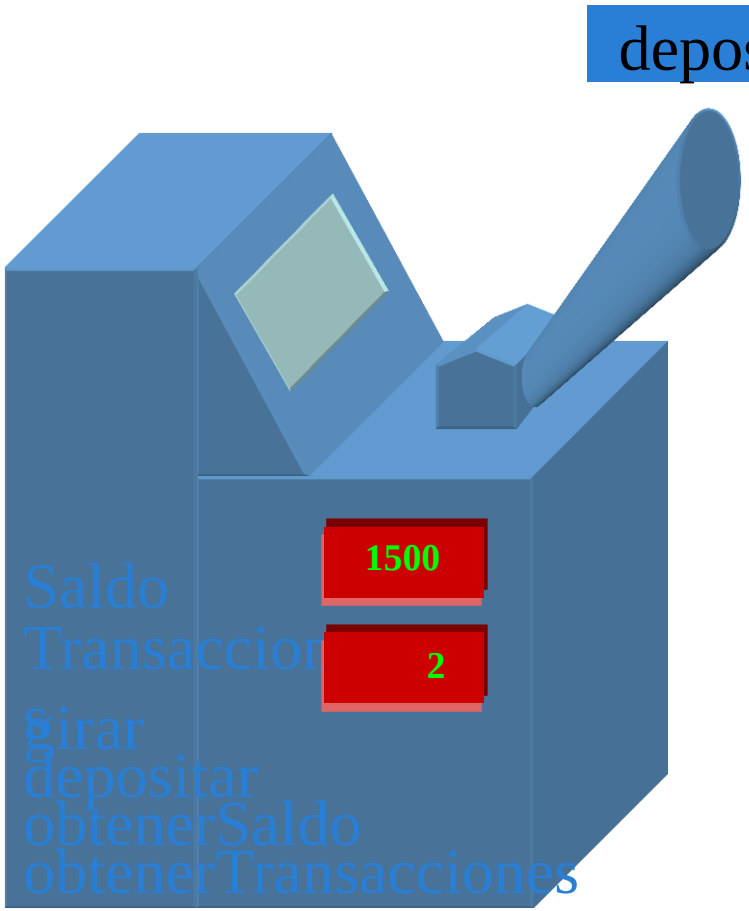
Opción Depositar
Monto $: 1000



Usuario

Aplicación

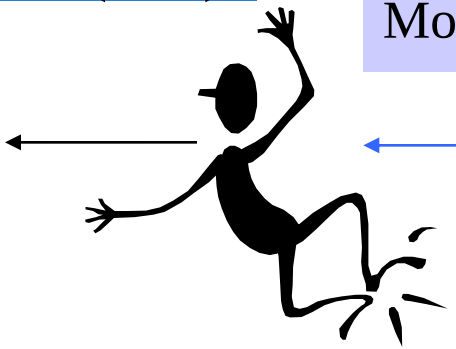
Ejemplo: depositar (otra vez)

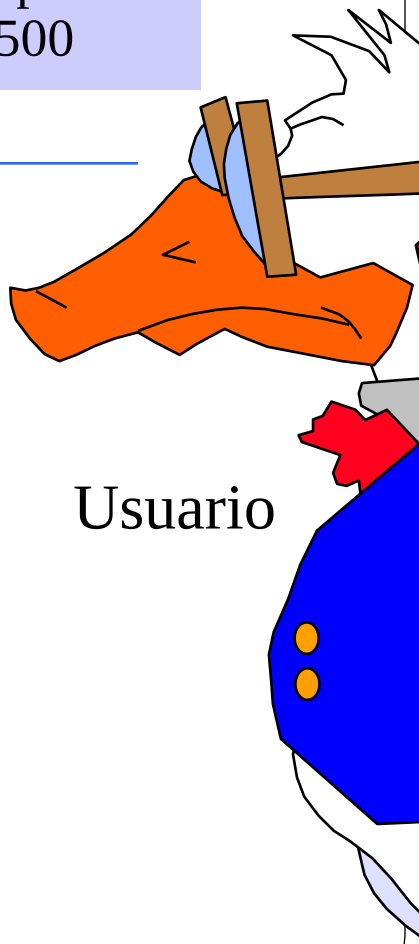


Objeto

depositar (500)

Opción Depositar
Monto $: 500

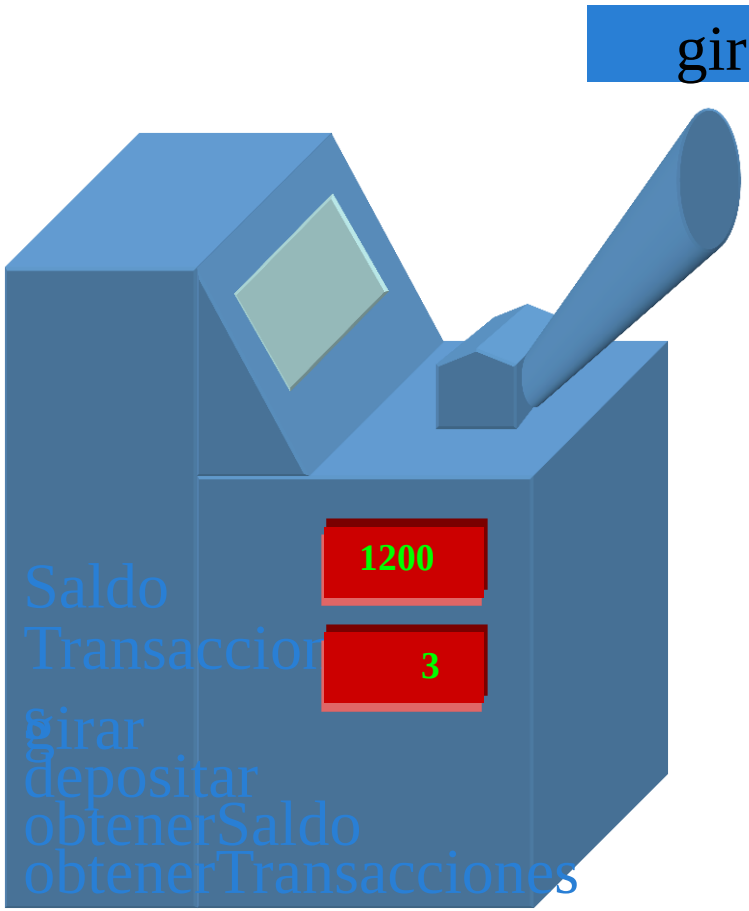




Usuario

Aplicación

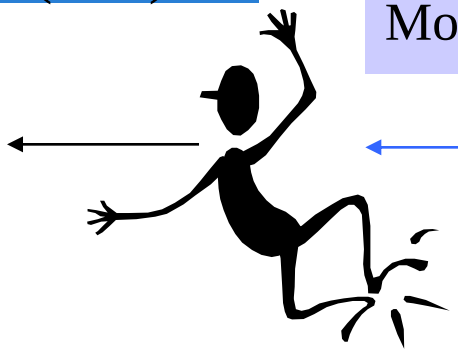
Ejemplo: girar

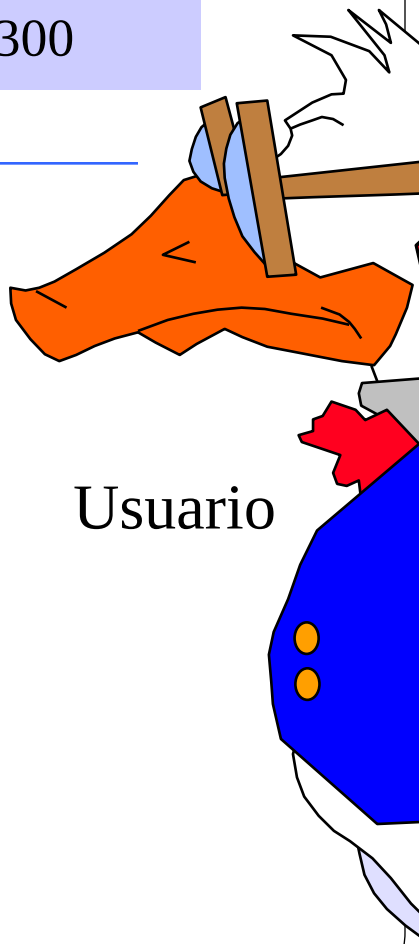


Objeto

girar (300)

Opción Girar
Monto $: 300





Usuario

Aplicación

Ejemplo: consultar estado