Programación Orientada a Objetos

3º de I.T.I.S
Curso 2001-2002

Francisco José García Peñalvo

Departamento de Informática y Automática - Universidad de Salamanca

Tema 1

Lenguajes de Programación Orientados a Objetos

Versión 3.0 – Febrero de 2002

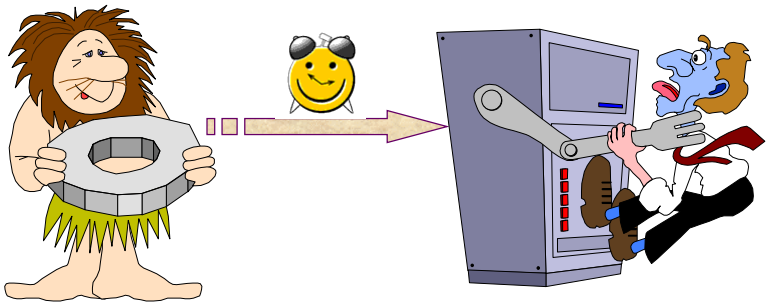
Departamento de Informática y Automática - Universidad de Salamanca

Esquema

1. Historia y evolución
2. Clasificación
3. Características
4. Revisión general de algunos LPOO representativos
5. Referencias
6. Lecturas complementarias

Departamento de Informática y Automática - Universidad de Salamanca 3

1. Historia y Evolución



Departamento de Informática y Automática - Universidad de Salamanca 4

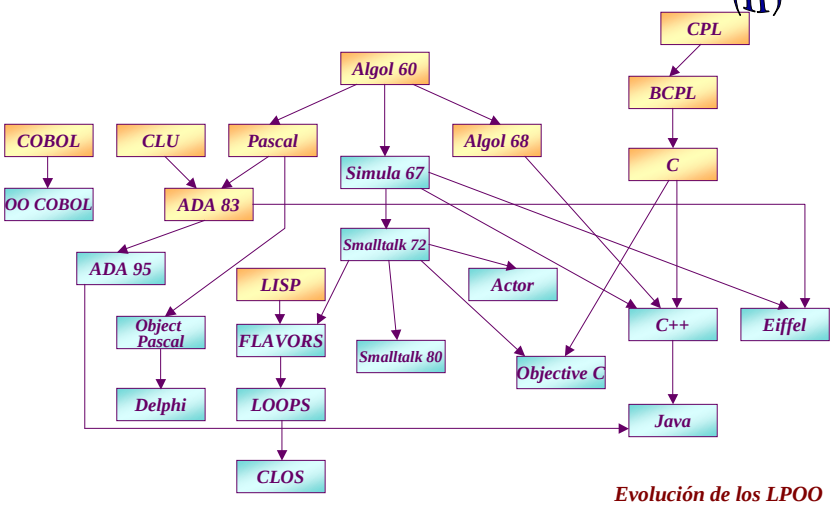
1.1 Historia de los LPOO (I)

1966 — Simula
1970 — Smalltalk 72
1980 — Smalltalk 80, C++, Objective-C, Eiffel
1990 — OO COBOL, ADA 95, SQL3, Delphi, Java

- *La historia de los LPOO comienza a finales de los años 60, con el lenguaje Simula, diseñado en el **Norwegian Computing Centre***
- *Durante la década de los 70s la poca actividad en el tema la monopoliza **Smalltalk***
- *La década de los 80s supone un nuevo auge para la POO*
- *Entre 1980 y 1983 Stroustrup desarrolla C++*
- *En la década de los 90s se produce la explosión total de las tecnologías orientadas al objeto*

Departamento de Informática y Automática - Universidad de Salamanca 5

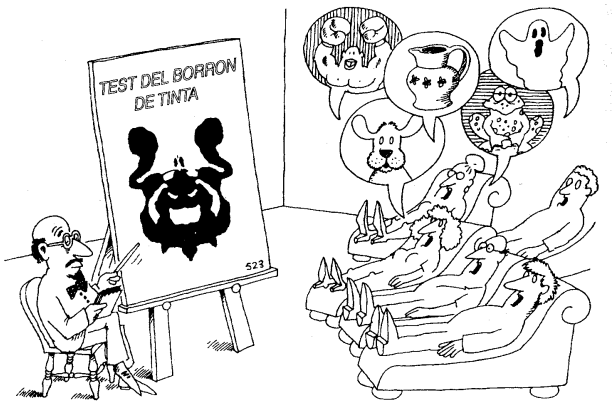
1.1 Historia de los LPOO (II)



Evolución de los LPOO

Departamento de Informática y Automática - Universidad de Salamanca 6

2. Clasificación de los LPOO



2.1 Introducción

Existen varias clasificaciones de los LPOO, atendiendo a criterios de construcción o características específicas de los mismos

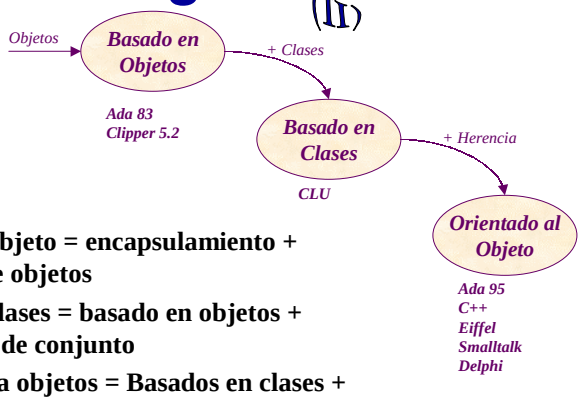
- *Clasificación de Peter Wegner*
- *Clasificación de Tesler*
- *Clasificación según el origen*

2.2 Clasificación de Wegner (I)

- ***Lenguajes Basados en Objetos:*** *Soportan objetos. Es decir, disponen de componentes caracterizados por un conjunto de operaciones (comportamiento) y un estado*
- ***Lenguajes Basados en Clases:*** *Disponen, además de objetos, de componentes tipo clase con operaciones y estado común*
- ***Lenguajes Orientados al Objeto:*** *Además de objetos y clases ofrecen mecanismos de herencia*

[Wegner, 1987]

2.2 Clasificación de Wegner (II)



- **Basado en objeto = encapsulamiento + identidad de objetos**
- **Basado en clases = basado en objetos + abstracción de conjunto**
- **Orientados a objetos = Basados en clases + herencia + autorrecursión**

2.3 Clasificación de Tesler (I)

	Orientados al proceso	Orientados al objeto
Estático	*POSL* Process-Oriented Static Languages *Pascal, C*	*OOSL* Object-Oriented Static Languages *C++, Java*
Dinámico	*PODL* Process-Oriented Dinamic Languages *Lisp, Scheme*	*OODL* Object-Oriented Dinamic Languages *Smalltalk, Dylan*

[Tesler, 1993]

Departamento de Informática y Automática - Universidad de Salamanca 11

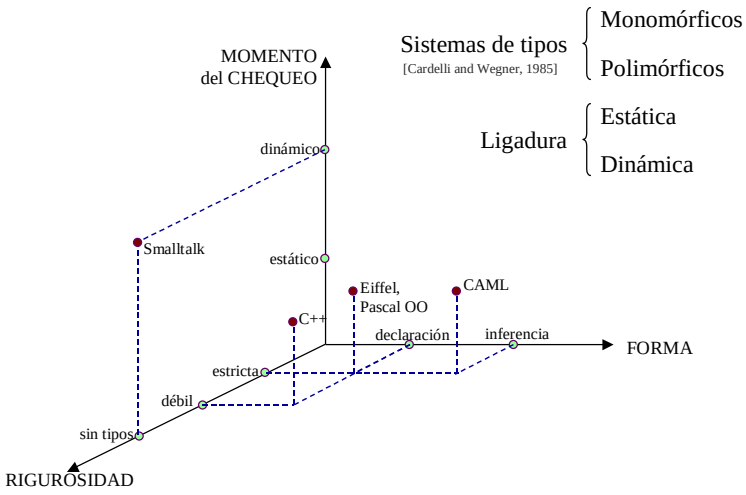
2.3 Clasificación de Tesler (II)

- Comprobación estricta
- Comprobación débil
- Sin tipos

Un sistema de tipos son las reglas que establecen la utilización de los tipos su corrección en un lenguaje de modelado o realización

Departamento de Informática y Automática - Universidad de Salamanca 12

2.3 Clasificación de Tesler (III)



2.3 Clasificación de Tesler (IV)

- Beneficios del uso de lenguajes con tipos estrictos
  - Evitan los fallos en tiempo de ejecución derivados de conflictos con los tipos
  - Detección temprana en tiempo de compilación de errores
  - Declaración de tipos como ayuda para la documentación
  - Eficiencia en el código

[Tesler, 1981]

2.3 Clasificación de Tesler (V)

Tipificación o tipado ("typing") es el proceso de declarar cuál es el tipo de información que puede contener una variable

Sistemas de tipos estáticos

- *También llamados sistemas con tipado fuerte o estricto*
- *Exigen asociación explícita de un tipo a cada nombre declarado*
- *El tipo de cada objeto se ha de determinar con anterioridad a la ejecución del programa*
- *Menos flexible pero más segura*
- *Ejecución más eficiente al no tener que hacer comprobación de tipos en tiempo de ejecución*

Sistemas de tipos dinámicos

- *También llamados sistemas con tipado débil,no estricto o dinámico*
- *No exigen asociación explícita de un tipo a cada variable*
- *Cada objeto conoce su tipo cuando se crea durante la ejecución*
- *Más flexibilidad*
- *Pérdida de eficiencia durante la ejecución del programa, debida a la necesidad de mantener y comprobar el tipo de todos los objetos durante la ejecución*

Departamento de Informática y Automática - Universidad de Salamanca 15

2.3 Clasificación de Tesler (VI)

Ligadura es el proceso de asociar un atributo a un nombre. En el caso de las funciones, el término ligadura (binding) se refiere a la conexión o enlace entre una llamada a función y el código real ejecutado como resultado de la llamada [Joyanes, 1998]

Departamento de Informática y Automática - Universidad de Salamanca 16

2.3 Clasificación de Tesler

- Ligadura estática o temprana
  - Resuelve las correspondencias en tiempo de compilación
  - No hay sobrecarga en tiempo de ejecución y los fallos se detectan en tiempo de compilación
  - Los enlaces no pueden hacerse sin volver a compilar
- Ligadura dinámica o tardía
  - La correspondencia se hace al invocar los métodos
  - Soporta la evolución de los programas sin tener que recompilar
  - Coste en tiempo de ejecución

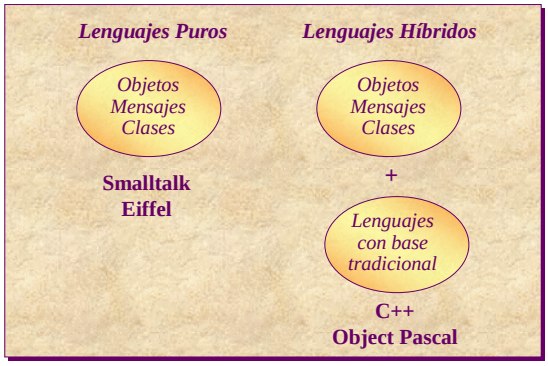
Departamento de Informática y Automática - Universidad de Salamanca 17

2.3 Clasificación de Tesler

	Comprobación Estática	***Comprobación Dinámica***
Ligadura Estática	*Garantía de corrección, interpretación inflexible* *C, Cobol*	*Combinación no válida (Sin sentido)*
Ligadura Dinámica	*Garantía de corrección, Interpretación flexible* *C++, Object Pascal*	*No hay garantía de corrección, interpretación flexible* *Smalltalk*

Departamento de Informática y Automática - Universidad de Salamanca 18

2.3 Clasificación según el Origen (I)



Departamento de Informática y Automática - Universidad de Salamanca 19

2.3 Clasificación según el Origen (II)

Lenguajes Puros

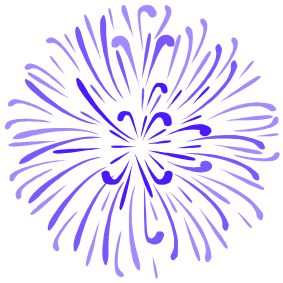
Ventajas	Desventajas
• Obligan a seguir los principios del paradigma objetual • Proporcionan su máxima flexibilidad para cambiar los aspectos esenciales del lenguaje	• En general, son menos rápidos que los híbridos • Son difíciles de codificar en toda clase de operaciones fundamentales • Pérdida de eficiencia en tiempo de ejecución

Lenguajes Híbridos

Ventajas	Desventajas
• Rápidos • Permiten una mayor integración con el entorno existente • Los programadores pueden seguir siendo productivos mientras aprenden las extensiones orientadas al objeto	• Pierden con frecuencia alguna característica del paradigma objetual • No permiten modificar las características de construcción del lenguaje base

Departamento de Informática y Automática - Universidad de Salamanca 20

3. Características de los LPOO



3.1 Elementos Concretos

- *Objetos*
- *Clases*
- *Herencia simple*
- *Herencia múltiple*
- *Comprobación estricta de tipos frente a comprobación débil*
- *Encapsulamiento*
- *Empaquetamiento*
- *Genericidad*
- *Paso de mensajes*
- *Polimorfismo*
- *Excepciones*
- *Concurrencia*
- *Persistencia*
- *Metadatos*
- *Afirmaciones y restricciones*
- *Bibliotecas de clases*
- *Eficiencia*
- *Administración de memoria*
- *Entorno de desarrollo*

[Rumbaugh et al., 1991]

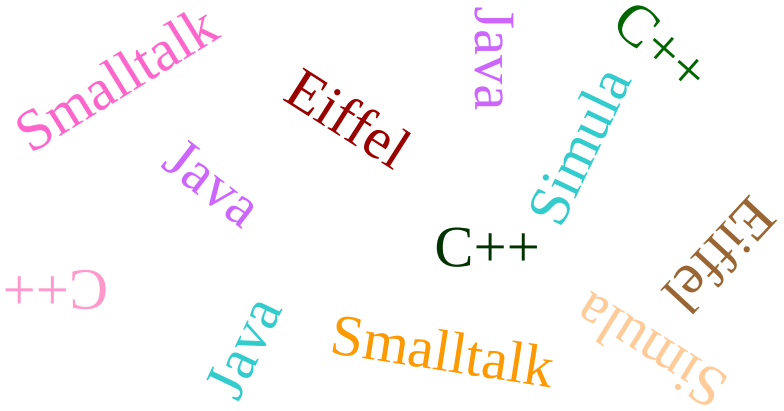
3.2 Requisitos Deseables en los LPOO

- Conceptos claros
- Orientación al Objeto pura
- Seguridad
- Alto nivel
- Modelo de ejecución simple
- Sintaxis fácil de leer
- Eliminación de la redundancia
- Pequeño
- De fácil transición a otros lenguajes
- Soporte para el aseguramiento de la corrección
- Entornos agradables

[Kölling, 1999]

Departamento de Informática y Automática - Universidad de Salamanca 23

4. Revisión de algunos de los Principales LPOO



Departamento de Informática y Automática - Universidad de Salamanca 24

4.1 Simula (I)

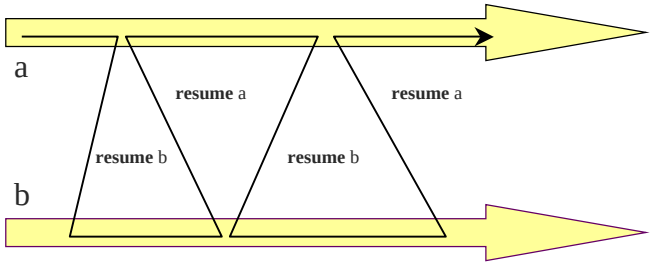
- El Simula 67 fue considerado como el primer LPOO
- Fue diseñado en 1967 por **Ole-Johan Dhal** y **Kristen Nygaard** en el *Norwegian Computing Center en Oslo*
- El nombre de Simula 67 fue acortado a Simula en 1986, existiendo un estándar del lenguaje desde 1987 [SIS, 1987]

4.1 Simula (II)

```
class POLIGONO;
        virtual: procedure set_vertices
begin
  procedure dibuja begin ... end;
end POLIGONO

POLIGONO class TRIANGULO
...
end TRIANGULO

...
ref(TRIANGULO) t; ref(POLIGONO) p;
...
p :- t;

p.set_vertices;

p.dibuja;

(p qua TRIANGULO).dibuja
```

4.1 Simula (III)

CORRUTINAS



4.1 Simula (IV)

CORRUTINAS

```
begin
class CONTROLLER; begin
        integer i;
   detach;
   for i:= 1 step 1 until 1000 resume printer
end CONTROLLER;

class PRINTER; begin
        integer i;
  detach;
  while true do
    for i:=1 step 1 until 8 do begin
      resume producer;
      outreal(producer.last_input);
      resume controller;
    end;
    next_line
  end
end PRINTER; ...
ref(PRINTER) printer; ref(CONTROLLER) controller; ...
printer :- new PRINTER; controller :- new CONTROLLER;...
resume controller
end
```

4.2 Smalltalk (I)

- Primer lenguaje creado con tecnología de objetos puros
- El origen de Smalltalk está alrededor de 1970 en la Universidad de Utah de la mano de **Alan Kay**
- Smalltalk se desarrolla en el **Xerox Palo Alto Research Center (PARC)** de la mano de **Alan Kay, Adele Goldberg** y **Daniel Ingalls**
- Los conceptos de Smalltalk han influido en el diseño de LPOO y en el aspecto y sensación de los GUI como los de Macintosh o Motif
- Smalltalk 72, Smalltalk 74, Smalltalk 76, Smalltalk 78 y Smalltalk 80.
- Smalltalk/V de Digital
- Entorno de programación: *Lenguaje Smalltalk básico*, *Colección de clases* y el *entorno real de programación*
- El estilo del lenguaje se enfatiza la **ligadura dinámica** y no realiza chequeo de tipos
- El bloque fundamental es la clase, que contiene la descripción de variables y métodos
- Los métodos contienen el código, y definen cómo responde un objeto a un mensaje

4.2 Smalltalk (II)

- La ejecución de una rutina de un objeto se denomina en la terminología de Smalltalk "*enviar un mensaje*" al objeto, cuya clase encontrará la forma apropiada de manejar dicho mensaje
- Todo en Smalltalk son objetos, incluyendo a las propias clases
- El entorno de Smalltalk permite un rápido desarrollo de programas
- La biblioteca de clases fue diseñada para ser extendida y adaptada por adición de subclases para satisfacer las necesidades de la aplicación
- La arquitectura Modelo/Vista/Controlador (**MVC**) es una importante contribución de Smalltalk
- Una referencia clásica sobre Smalltalk es [Goldberg and Robson, 1983]

4.2 Smalltalk (III)

Abstracción	*Variables de instancia* *Métodos de instancia* *Variables de clase* *Métodos de clase*	Sí Sí Sí Sí
Encapsulamiento	*De variables* *De métodos*	Privado Público
Modularidad	*Tipos de módulo*	Ninguno
Jerarquía	*Herencia* *Unidades genéricas* *Metaclases*	Simple No Sí
Tipos	*Comprobación estricta* *Polimorfismo*	No Sí
Concurrencia	*Multitarea*	Indirectamente (mediante clases)
Persistencia	*Objetos persistentes*	No

Departamento de Informática y Automática - Universidad de Salamanca 31

4.3 C++ (I)

- C++ es un lenguaje híbrido, extensión del lenguaje C
- Fue desarrollado por **Bjarne Stroustrup** en los laboratorios AT&T
- Las primeras versiones del lenguaje se denominaron **C con clases**, y datan de 1980
- El nombre de C++ se debe a **Rick Mascitti**, quien lo bautizó de esta manera en el verano de 1983
- C++ fue instalado por primera vez fuera del grupo de investigación del autor en julio de 1983
- En 1987 surge la necesidad de estandarización de C++
- Dispone de capacidades para la herencia simple y múltiple
- Por defecto C++ tiene ligadura estática
- Las funciones definidas como virtuales pueden beneficiarse del concepto de ligadura dinámica

Departamento de Informática y Automática - Universidad de Salamanca 32

4.3 C++ (III)

- C++ es estrictamente tipado, aunque admite la posibilidad de casting.
- No cuenta con un *recolector de basura*
- La ausencia de un gestor automático de memoria por defecto, obliga al concepto de destructor
- Ocultamiento de la información, incluso a los descendientes
- Cuenta con un mecanismo de acceso especial por parte de una clase o una función que se declare como *friend*
- Soporte de manejo de excepciones
- Soporte de genericidad mediante las plantillas
- Permite sobrecarga de operadores
- Sintaxis oscura, y gramática difícil de analizar
- Lenguaje complejo, preocupado por la detección temprana de errores, y por la eficiencia de la ejecución, perdiendo algo de sencillez y de flexibilidad para el diseño
- [Stroustrup, 1997] es una referencia válida para este lenguaje

4.3 C++ (III)

```
#include <iostream.h>
#include <string.h>

class Saludos {
        char *saludo;
public:
        Saludos(char *msg) {
                int longitud = strlen(msg);
                saludo = new char[longitud+1];
                strcpy(saludo, msg);
        }

        char* muestra(void) {return saludo;}
};

void main(void) {
        Saludos s1("Hola Mundo }:()");
        Saludos s2("Bye, dear friends :((");

        cout << s1.muestra() << endl;
        cout << s2.muestra() << endl;
}
```

4.3 C++ (IV)

Abstracción	*Variables de instancia* *Métodos de instancia* *Variables de clase* *Métodos de clase*	Sí Sí Sí Sí
Encapsulamiento	*De variables* *De métodos*	Público, Privado, Protegido Público, Privado, Protegido
Modularidad	*Tipos de módulo*	Fichero
Jerarquía	*Herencia* *Unidades genéricas* *Metaclases*	Múltiple Sí No
Tipos	*Comprobación estricta* *Polimorfismo*	Sí Sí
Concurrencia	*Multitarea*	Indirectamente (mediante clases)
Persistencia	*Objetos persistentes*	No

Departamento de Informática y Automática - Universidad de Salamanca 35

4.4 Eiffel (I)

- Eiffel fue desarrollado por **Bertrand Meyer**
- Es un lenguaje orientado al objeto puro
- Soporta ligadura dinámica
- Tiene comprobación estricta de tipos
- Admite herencia múltiple
- Soporta clases parametrizadas
- La gestión de memoria la lleva a cabo el entorno de programación
- Eiffel proporciona una biblioteca de clases predefinidas muy completa
- Eiffel cuenta con la clase como único criterio de estructuración
- Una declaración de clase en Eiffel puede incluir una lista de características exportadas, una lista de clases predecesoras, y una lista de declaraciones de características
- Cuenta con manejo de excepciones
- [Meyer, 1992] es una de las referencias obligadas, aunque con unas miras más amplias, se recomienda [Meyer, 1997]

Departamento de Informática y Automática - Universidad de Salamanca 36

4.4 Eiffel

- ***Diseño por contrato***

Aserciones
- *Precondiciones: que deben cumplirse antes de que se invoque un método*
- *Postcondiciones: que se garantizan han de cumplirse después de que se haya ejecutado el método*
- *Invariantes: condición que debe cumplir la clase en toda circunstancia estable*

- Un contrato entre el proveedor y el usuario de una clase espera que
  - El usuario de una clase asegurará que las precondiciones de un método se cumplan antes de que el método se invoque
  - El proveedor de la clase garantizará que las postcondiciones se cumplan después que se ha aplicado el método

Departamento de Informática y Automática - Universidad de Salamanca 37

4.4 Eiffel

- ***Violación de Aserciones***

¿De quién es la responsabilidad?
- *Precondiciones: De quien solicita el servicio (cliente)*
- *Postcondiciones: De quien ofrece el servicio (servidor)*
- *Invariantes: De quien esté actuando en ese momento sobre el objeto*

- *¿Qué provoca la violación de una aserción? Excepciones*

- ¡Qué se encargue el responsable!
  - Manejo de la excepción: tratar de arreglar la situación, cláusula de rescate: `rescue` , intentar nuevamente: `retry`
  - Pasar la responsabilidad... ¿Nadie? El sistema aborta

Aserciones y Herencia: La subcontratación
el subcontratado pide menos y garantiza más (sino no es negocio ☺)

Departamento de Informática y Automática - Universidad de Salamanca 38

4.4 Eiffel (IV)

```
class STACK[T]
creation
   create
feature {NONE}
   elems : ARRAY[T];
feature
   max_size : INTEGER;
   total : INTEGER;
   create(n : INTEGER) is
   require
      n > 0
   do
      !!elems.make(1,n);
      max_size := n
   ensure
      max_size = n;
      total = 0;
   end;
   empty : BOOLEAN is do
      Result := total = 0
   ensure
      equal(strip(), old strip())
   end;
   full : BOOLEAN is do
      Result := total =
      max_size
   ensure
      equal(strip(), old strip())
   end;
   top : T is require
      not empty
   do
      Result := elems@total
   ensure
      equal(strip(), old strip())
   end
   push(x : T) is require
      not full
   do
      total := total + 1;
      elems.put(x,total)
   ensure
      not empty;
      total = old total + 1;
      top = x
   end;
   pop is require
      not empty
   do
      total := total -1
   ensure
      total = old total -1;
      not full
   end
invariant
   0 <= total;
   total <= max_size
end – STACK
```

Departamento de Informática y Automática - Universidad de Salamanca 39

4.4 Eiffel (V)

Abstracción	*Variables de instancia* *Métodos de instancia* *Variables de clase* *Métodos de clase*	Sí Sí No No
Encapsulamiento	*De variables* *De métodos*	Público (sólo lectura), Privado, Selectivamente Público, Privado, Selectivamente
Modularidad	*Tipos de módulo*	Clase
Jerarquía	*Herencia* *Unidades genéricas* *Metaclases*	Múltiple Sí No
Tipos	*Comprobación estricta* *Polimorfismo*	Sí Sí
Concurrencia	*Multitarea*	Si (mediante clases de bibliotecas)
Persistencia	*Objetos persistentes*	Si (mediante clases)

Departamento de Informática y Automática - Universidad de Salamanca 40

4.5 Java (I)

- Realizado por un equipo de **Sun Microsystems** a finales de 1995
- Ha recibido una especial atención desde los primeros meses de 1996
- Es un lenguaje orientado al objeto puro diseñado desde cero, que recibe muchas influencias de C++
- Se le atribuyen las siguientes características
  - *Simple y poderoso*
  - *Seguro*
  - *Robusto*
  - *Interactivo*
  - *Independiente de la arquitectura*
  - *Interpretado*
  - *Sencillo de aprender*
- Soporta *threads*
- *Recogida de basura* automática
- De la abundante bibliografía existente se recomienda [SUN, 2001]

Departamento de Informática y Automática - Universidad de Salamanca 41

4.5 Java (II)

- Java produce un **bytecode** que será interpretado por una **máquina virtual**
- La máquina virtual se encuentra a menudo en navegadores web
- La explosión de Internet ha influido en el auge de Java

Java es un componente principal dentro del currículo de la Ingeniería del Software varias Universidades

En los primeros seis meses de 1996, Java apareció en la prensa de US 4325 veces, como punto de comparación **Bill Gates** apareció 5096 veces

Departamento de Informática y Automática - Universidad de Salamanca 42

4.5 Java (III)

```
class _saludos {
        String saludo="";

        _saludos(String msg) {
                saludo = msg;
        }

        public void mostrar() {
                System.out.println(saludo);
        }
}

class Saludos {
        public static void main(String args[]) {
                _saludos s1 = new _saludos("Hola Mundo :)))");
                _saludos s2 = new _saludos("Adios amigos :((");

                s1.mostrar();
                s2.mostrar();
         }
}
```

4.5 Java (IV)

Abstracción	*Variables de instancia* *Métodos de instancia* *Variables de clase* *Métodos de clase*	Sí Sí Sí Sí
Encapsulamiento	*De variables* *De métodos*	Público, Privado, Protegido Público, Privado, Protegido
Modularidad	*Tipos de módulo*	Fichero
Jerarquía	*Herencia* *Unidades genéricas* *Metaclases*	Simple No No
Tipos	*Comprobación estricta* *Polimorfismo*	Sí Sí
Concurrencia	*Multitarea*	Sí
Persistencia	*Objetos persistentes*	No

5. Referencias

[Cardelli and Wegner, 1985] Cardelli, L., Wegner, P. "*On Understanding Types, Data Abstraction and Polymorphism*". ACM Computing Surveys, 17(4). 1985.

[Goldberg and Robson, 1983] Goldberg, A., Robson, D. "*Smalltalk-80: The Language and its Implementation*". Addison-Wesley, 1983.

[Joyanes, 1998] Joyanes Aguilar, L. "*Programación Orientada a Objetos*". 2ª Edición. McGraw-Hill, 1998.

[Kölling, 1999] Kölling, M. "*The Problem of Teaching Object-Oriented Programming, Part 1: Languages*". Journal of Object-Oriented Programming, 11(8):8-15. January, 1999.

[Meyer, 1992] Meyer, B. "*Eiffel: The Language*". Prentice Hall Object-Oriented Series, 1991; second revised printing, 1992.

[Meyer, 1997] Meyer, B. "*Object Oriented Software Construction*". 2nd Edition. Prentice Hall, 1997.

[Rumbaugh et al., 1991] Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., Lorensen, W. "*Object-Oriented Modeling and Design*". Prentice-Hall, 1991.

[SIS, 1987] SIS. "*Data Processing - Programming Languages — SIMULA*". Standardiseringskommissionen i Sverige (Swedish Standards Institute), Svensk Standard SS 63 61 14, 20 May, 1987.

[Stroustrup, 1997] Stroustrup, B. "*The C++ Programming Language*". 3rd Edition, Addison Wesley, 1997.

[SUN, 2001] SUN Microsystems. "*The Java Tutorial. A Practical Guide for Programmers*". http://java.sun.com/docs/books/tutorial/index.html. [Última vez visitado, 13/2/2002]. December 2001.

[Tesler, 1981] Tesler, L. "*The Smalltalk Environment*". Byte, 6(8). August, 1981.

[Tesler, 1993] Tesler, L. "*Object-Oriented Dynamic Languages*". In Proceedings of the Object Expo Conference. July, 1993.

[Wegner, 1987] Wegner, P. "*The Object-Oriented Classification Paradigm in Research Directions on Object-Oriented Programming*". MIT Press, Cambridge, MA, 1987.

6. Lecturas Complementarias (1)

"*The Real Stroustrup Interview*". IEEE Computer, 31(6):110-114. June, 1998.

En 1998 se difundió por Internet una supuesta entrevista con Bjarne Stroustrup en la prácticamente el autor de C++ tiraba por los suelos su obra. Así en el mes de junio de este mismo año se publica en la revista IEEE Computer una entrevista, esta vez de verdad con Stroustrup, donde da su opinión sobre el papel de C++ en el mundo de la Programación Orientada a Objetos.

Interactive Software Engineering. "*Object-Oriented Languages: A Comparison*". **Interactive Software Engineering (ISE)**. http://www.eiffel.com/doc/manuals/technology/oo_comparison/page.html [Última vez visitado, 13-2-2002]. 2001.

Tablas comparativas entre Eiffel, C++, Java y Smalltalk.

Joyner, I. "*C++?? A Critique of C++ and Programming and Language Trends of the 1990s*". 3rd edition http://www.progsoc.uts.edu.au/~geldridg/cpp/cppcv3/cppcv3.pdf. [Última vez visitado 7/1/2000]. 1996.

Informe crítico sobre C++, comparando sus características con otros lenguajes de programación, en especial con Eiffel.

Kay, A C. "*The Early History of Smalltalk*". In Proceedings of the second ACM SIGPLAN conference on History of programming languages – HOPL II. (April 20 - 23, 1993, Cambridge United States). ACM SIGPLAN Notices, 28(3):69-75. March, 1993.

Artículo sobre los orígenes de Smalltalk.

Meyer, B. "*Approaches to Portability*". Journal of Object-Oriented Programming (JOOP), 11(4):68-70. July/August, 1998.

Comparativa de los bytecodes de Java con el método propio de Eiffel consistente en utilizar C como lenguaje intermedio para conseguir la portabilidad del software.

6. Lecturas Complementarias (II)

Meyer, B. "*Construcción de Software Orientado a Objetos*". 2ª Edición. Prentice Hall, 1999.

En relación con el presente tema se destaca en capítulo 35, **De Simula a Java y más allá: los principales entornos y lenguajes O-O**, donde se repasan algunos de los lenguajes de programación orientados a objetos de mayor peso.

Piattini, M. G. "*Selección de Lenguajes de Programación Orientados al Objeto: ¿Cuestión de Religión?*". Revista BASE de la ALI (Asociación de Doctores, Licenciados e Ingenieros en Informática), Nº24:58-62. Abril, 1994.

Artículo intoductorio al mundo de los LPOO.

Prechelt, L. "*Comparing Java vs. C/C++ Efficiency Differences to Interpersonal Differences*". Communications of the ACM, 42(10):109-112. October 1999.

Artículo en el que se presenta un experimento llevado a cabo para comparar la eficiencia de estos lenguajes de programación.

Rans, M. "*A History of Object-Oriented Programming Languages and their Impact on Program Design and Software Development*". http://jeffsutherland.com/papers/Rans/OOlanguages.pdf [Última vez visitado, 13/2/2002]. November 1999.

Resumen de la historia de algunos lenguajes de programación orientados a objetos destacados.

Stroustrup, B. "*The Design and Evolution of C++*". Addison-Wesley, 1994. Reprinted with corrections in April 1995.

Libro del autor de C++ explicando la historia, diseño y evolución de C++.