ST0244 Lenguajes de Programacion
4. Programación orientada a objetos

Andrés Sicard Ramírez

Universidad EAFIT

Semestre 2024-1

Preliminares

Convenciones

- La numeración (capítulos, teoremas, figuras, páginas, etc) en estas diapositivas corresponde a la numeración del texto guía [Lee 2017].
- Los ejemplos que incluyen código fuente están en el repositorio del curso.

Introducción

Concepto		Cuenta	%
Herencia	(*Inheritance*)	71	81 %
Objeto	(*Object*)	69	78 %
Clase	(*Class*)	62	71 %
Encapsulación	(*Encapsulation*)	55	63 %
Método	(*Method*)	50	57 %
Paso de mensajes	(*Message passing*)	49	56 %
Polimorfismo	(*Polymorphism*)	47	53 %

«*The quarks of object-oriented development* » [Armstrong 2006].

Introducción

Definición de los *quarks* [Armstrong 2006]

(i) «***Inheritance:*** *A mechanism that allows the data and behavior of one class to be included in or used as the basis for another class.*»

(ii) «***Object:*** *Individual, identifiable item, either real or abstract, which contains data about itself and descriptions of its manipulations of the data.*»

(iii) «***Class:*** *A description of the organization and similar objects.*»

(iv) «***Encapsulation:*** *A technique for designing classes and objects that restricts access to the data and behavior by defining a limited set of messages that an object of that class can receive.*»

(continua en la próxima diapositiva)

Introducción

Quark's definitions (continuation)

(v) «***Method:*** *A way to access, set or manipulate object's information.*»

(vi) «***Message passing:*** *The process by which an object sends data to another object or asks the other object to invoke a method.*»

(vii) «***Polymorphism*** *is defined as: the ability of different classes to respond to the same message and each implement the method appropriately.*»

Introducción

Lenguajes de programación

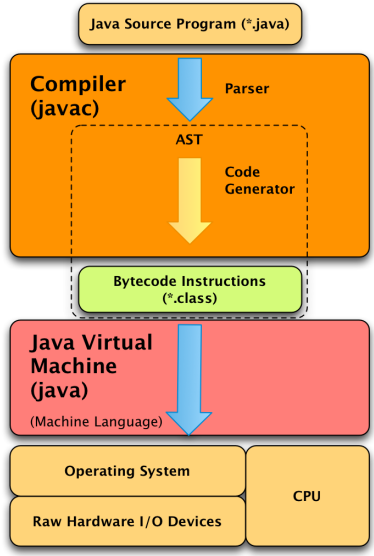
Algunos lenguajes orientados a objetos son C++, Java, Python, Ruby, Scala y Smalltalk.

El entorno de Java

Herramientas

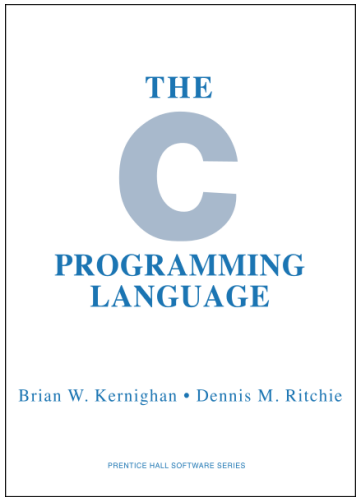
- Compilador (comando `javac` en Linux): Convierte un programa fuente en Java a *bytecode*.
- *Java Virtual Machine (JVM)* (comando `java` en Linux): Ejecuta el *bytecode*.

(Fig. 4.4)



El entorno de Java

Acerca del ejemplo «hello, world»



«*The first program to write is the same for all languages: Print the words* `hello, world`.» [1978, § 1.1]

El entorno de Java

Ejemplo

Determine la versión de Java que tiene instalada usando los siguientes comandos:

```
$ javac --version
javac 21.0.1
```

```
$ java --version
openjdk 21.0.1 2023-10-17
```

El entorno de Java

Ejemplo

Compile y ejecute el programa `hw/HelloWorld.java` usando los siguientes comandos:

```
$ git clone https://github.com/asr/st0244-pl.git
$ cd st0244-pl/hw
$ javac HelloWorld.java
$ java HelloWorld
Hello World! (Java)
```

El entorno de Java

Ejemplo

Compilando el programa `oop/Warning.java` usando la opción `-Xlintx`:

```
$ javac -Xlint Warning.java
Warning.java:4: warning: [cast] redundant cast to String
      String s = (String)"hello, word!";
                 ^
1 warning
```

El entorno de Java

Ejemplo

Compilando el programa `oop/Warning.java` usando las opciones `-Xlint` y `-Werror`:

```
$ javac -Xlint -Werror Warning.java
Warning.java:4: warning: [cast] redundant cast to String
      String s = (String)"hello, word!";
                 ^
error: warnings found and -Werror specified
1 error
1 warning
```

Ejemplo

La documentación a los argumentos de `-Xlint` se encuentra en:

`https://docs.oracle.com/en/java/javase/21/docs/specs/man/javac.html#examples-of-using--xlint-keys` .

El entorno de Java

Ejemplo

La documentación a los argumentos de `-Xlint` se encuentra en:

`https://docs.oracle.com/en/java/javase/21/docs/specs/man/javac.html#examples-of-using--xlint-keys` .

Observación

Para obtener información acerca de opciones adicionales y los argumentos usados con `-Xlint`, use los siguientes comandos respectivamente:

```
$ javac --help -X
$ javac --help-lint
```

El entorno de C++

El entorno de C++

Fig. 4.5:

https://kentdlee.github.io/PL/build/html/oop.html#the-c-environment

El entorno de C++

Ejemplo

En mi computador determino las versiones de C++ que tengo instaladas usando los siguientes comandos:

(i) GCC

```
$ g++ --version
g++ (Ubuntu 10.5.0.1ubuntu1~22.04) 10.5.0.1
```

```
$ g++-12 --version
g++-12 (Ubuntu 12.1.0-2ubuntu1~22.04) 12.1.0
```

(ii) Clang

```
$ clang++ --version
Ubuntu clang version 14.0.0-1ubuntu1.1
```

Ejemplo

Compile y ejecute el programa `hw/hello-world.cc` usando los siguientes comandos:

```
$ g++ -o hello-world hello-world.cc
$ ./hello-world
Hello World! (C++)
```

El entorno de C++

Ejemplo

Compilando el programa `oop/warning.cc` usando la opción `-Wall`:

```
$ g++ -Wall -o warning warning.cc
warnings.cc: In function 'int main(int, char**)':
warnings.cc:10:11: warning: 'i' is used uninitialized
in this function [-Wuninitialized]
cout << i << endl;
        ^
```

El entorno de C++

Ejemplo

Compilando el programa `oop/warning.cc` usando las opciones `-Wall` y `-Werror`:

```
$ g++ -Wall -Werror -o warning warning.cc
warnings.cc: In function 'int main(int, char**)':
warnings.cc:10:11: warning: 'i' is used uninitialized
in this function [-Wuninitialized]
cout << i << endl;
        ^
cc1plus: all warnings being treated as errors
```

El macro procesador para C++

Descripción

El **macro procesador** para C++ es un programa que procesa **directivas** (*directives*). Las directivas dan instrucciones (*p. ej.* inclusión de archivos, compilación condicional, definición y expansión de macros, entre otras) al compilador para preprocesar el programa fuente antes de que la compilación comience.

El macro procesador para C++

Descripción

El **macro procesador** para C++ es un programa que procesa **directivas** (*directives*). Las directivas dan instrucciones (*p. ej.* inclusión de archivos, compilación condicional, definición y expansión de macros, entre otras) al compilador para preprocesar el programa fuente antes de que la compilación comience.

Ejemplo

Debido a la línea

```
#include <iostream>
```

en el archivo `hw/hello-world.cc`, el macro procesador incluye la biblioteca `iostream`.

El macro procesador para C++

Observación

El macro procesador para C++ es el programa cpp.

Ejemplo

Usando cpp (o de forma equivalente, g++ con la opción `-E`):

```
$ cpp hello-world.cc > xxx.cc
```

Visualice el archivo `xxx.cc` en su editor favorito.

¿Cuántas lineas tienen el archivo `xxx.cc`?

```
$ wc -l xxx.cc
28647 xxx.cc
```

La herramienta make

Descripción

«The make tool is a program that can be used to compile programs that are composed of modules and utilise separate compilation.» (pág. 120)

La herramienta make

Descripción

> *«The make tool is a program that can be used to compile programs that are composed of modules and utilise separate compilation.» (pág. 120)*

Reglas

Un `Makefile` es un archivo que procesa el programa make. Este archivo contiene un conjunto de reglas de la siguiente forma:

```
target ... : prerequisites ...
        recipe
        ...
```

La herramienta make

Ejemplo (regla para el make)

```
foo : foo.cc
        g++ -o foo foo.cc
```

La herramienta make

Ejemplo

Mirar el archivo `hw/Makefile`.

Clases y objetos

Descripción

*«Object-Oriented programming is all about creating objects. Objects have **state information**, sometimes just called **state**, and **methods** that operate on that state, sometimes altering the state. . . . A **class** defines the state information maintained by an object and the methods that operate on that state.» (pág. 127)*

Clases y objetos

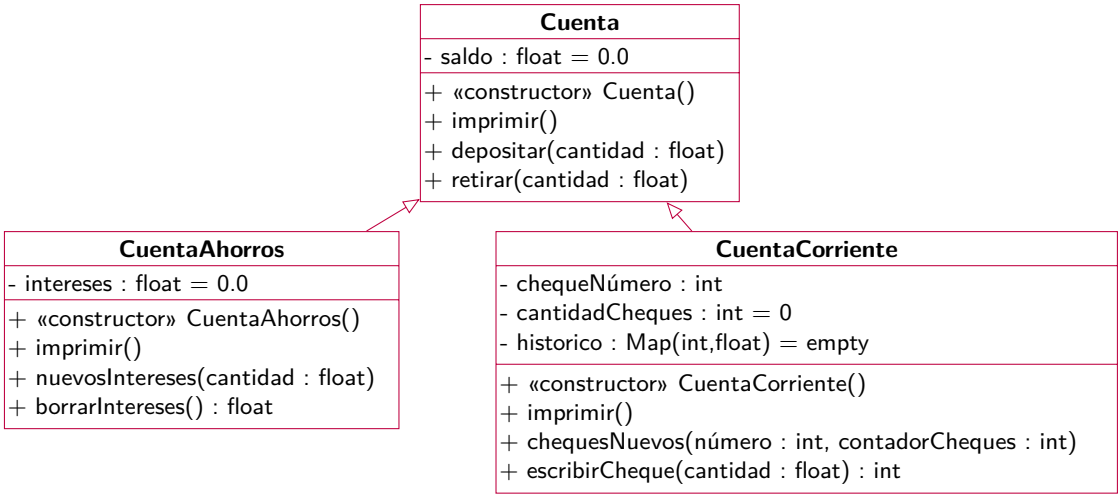
Ejemplo

El tutorial de GNU Smalltalk* describe la implementación de un sistema de cuentas bancarias simplificado empleando tres clases: Cuenta (*Account*), Cuenta de Ahorros (*Savings*) y Cuenta Corriente (*Cheking*).

*En https://www.gnu.org/software/smalltalk/manual/html_node/Tutorial.html.

Clases y objetos

Ejemplo (Diagrama UML de clases para el sistema de cuentas bancarias simplificado)



Clases y objetos

Ejemplo

Una implementación en Java del sistema de cuentas bancarias simplificado se encuentra en el directorio `oop/sistema-cuentas-bancarias/java`.

Para ejecutar el programa use los siguientes comandos:

```
$ cd oop/sistema-cuentas-bancarias/java
$ ant run
```

Clases y objetos

Ejemplo

Una implementación en C++ del sistema de cuentas bancarias simplificado se encuentra en el directorio `oop/sistema-cuentas-bancarias/c-plusplus`.

Para ejecutar el programa use los siguientes comandos:

```
$ cd oop/sistema-cuentas-bancarias/c-plusplus
$ make scb
```

Herencia y polimorfismo

Definición

*«**Inheritance** is a relationship among classes wherein one class shares the structure and/or behavior defined in one (**single inheritance**) or more (**multiple inheritance**) other classes. We call the class from which another class inherits its **superclass**... Similarly, we call a class that inherits from one or more classes a **subclass**.» (Booch, Maksimchuk, Engle, Young, Conallen y Houston 2007, pág. 100)*

Herencia y polimorfismo

Definición

*«**Inheritance** is a relationship among classes wherein one class shares the structure and/or behavior defined in one (**single inheritance**) or more (**multiple inheritance**) other classes. We call the class from which another class inherits its **superclass**... Similarly, we call a class that inherits from one or more classes a **subclass**.» (Booch, Maksimchuk, Engle, Young, Conallen y Houston 2007, pág. 100)*

*«**Inheritance** is: a mechanism that allows the data and behavior of one class to be included in or used as the basis for another class.» (Armstrong 2006, pág. 124)*

Herencia y polimorfismo

Definición

*«**Inheritance** is a relationship among classes wherein one class shares the structure and/or behavior defined in one (**single inheritance**) or more (**multiple inheritance**) other classes. We call the class from which another class inherits its **superclass**... Similarly, we call a class that inherits from one or more classes a **subclass**.» (Booch, Maksimchuk, Engle, Young, Conallen y Houston 2007, pág. 100)*

*«**Inheritance** is: a mechanism that allows the data and behavior of one class to be included in or used as the basis for another class.» (Armstrong 2006, pág. 124)*

*«**Inheritance** is the mechanism we employ to re-use code in software we are currently writing.» (pág. 131)*

Herencia y polimorfismo

Definición

*«**Polymorphism** is a concept in type theory wherein a name may denote instances of many different classes as long as they are related by some common superclass. . . With polymorphism, an operation can be implemented differently by the classes in the hierarchy. In this manner, a subclass can extend the capabilities of its superclass or override the parent's operation.» (Booch, Maksimchuk, Engle, Young, Conallen y Houston 2007, pág. 102)*

Herencia y polimorfismo

Definición

*«**Polymorphism** is a concept in type theory wherein a name may denote instances of many different classes as long as they are related by some common superclass. . . With polymorphism, an operation can be implemented differently by the classes in the hierarchy. In this manner, a subclass can extend the capabilities of its superclass or override the parent's operation.» (Booch, Maksimchuk, Engle, Young, Conallen y Houston 2007, pág. 102)*

*«**Polymorphism** is defined as: the ability of different classes to respond to the same message and each implement the method appropriately.» (Armstrong 2006, pág. 126)*

Herencia y polimorfismo

Definición

*«**Polymorphism** is a concept in type theory wherein a name may denote instances of many different classes as long as they are related by some common superclass. . . With polymorphism, an operation can be implemented differently by the classes in the hierarchy. In this manner, a subclass can extend the capabilities of its superclass or override the parent's operation.» (Booch, Maksimchuk, Engle, Young, Conallen y Houston 2007, pág. 102)*

*«**Polymorphism** is defined as: the ability of different classes to respond to the same message and each implement the method appropriately.» (Armstrong 2006, pág. 126)*

*«**Polymorphism** is the mechanism we employ to customize the behavior of code we have already written.» (pág. 131)*

Herencia y polimorfismo

Observación

El polimorfismo descrito es llamado **run-time polymorphism** o **subtype polymorphism**.

Herencia y polimorfismo

Ejemplos

Mirar los archivos `oop/Polymorphism.java` y `oop/polymorphism.cc`.

- Java

```
$ cd oop
$ make Polymorphism
$ java Polymorphism
```

- C++

```
$ cd oop
$ make polymorphism
$ ./polymorphism
```

Espacios de nombres

Descripción

En los lenguajes de programación un **espacio de nombres** (*namespaces*) es un contexto para los identificadores usados en el programa. Éstos ayudan a identificar los nombres de clases, funciones, métodos, variables, etc.

Espacios de nombres

Ejemplo (C++)

La línea

```
using namespace std;
```

en el archivo `hw/hello-world.cc` abre el espacio de nombre `std` (standard). Si ésta es removida, se debería reemplazar la línea

```
cout << "Hello␣World!" << endl;
```

por la línea

```
std::cout << "Hello␣World!" << std::endl;
```

donde '`::`' es calificador de alcance (*scope qualifier*).

Espacios de nombres

Ejemplo (Java)

Los espacios de nombres en Java son manejados por paquetes (colección de clases bajo un nombre)

- A partir de la línea

```
import java.io.File;
```

  podemos escribir `File` en lugar de `java.io.File`.

- A partir de la línea

```
import java.io.*;
```

  no necesitamos calificar los nombres cuando se usan las clases en `java.io`.

Espacios de nombres

Observación

«The safest way to program is to not open up namespaces or merge them together. But, that is also inconvenient since the whole name must be written each time. What is correct for your program depends on the program being written.» (pág. 122)

Enlazamiento

Enlazamiento de bibliotecas

Las bibliotecas son comunes y necesarias en los lenguajes de programación. Para que éstas puedan ser usadas es necesario enlazarlas (*linking*) a los programas.

Enlazamiento

Enlazamiento estático y enlazamiento dinámico*

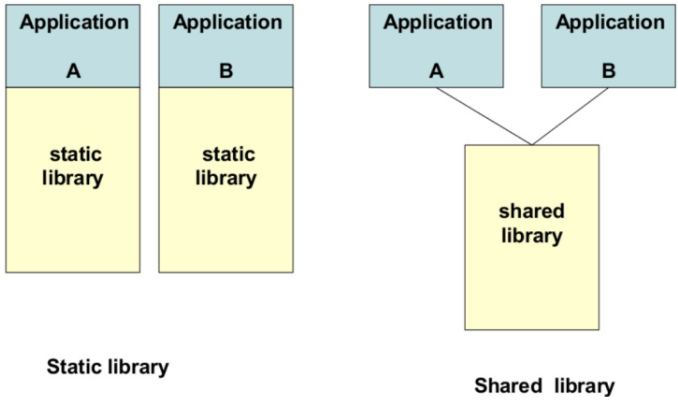


*Figura tomada de
https://achindrabhatnagar.wordpress.com/2018/09/08/static-and-dynamic-linking-c-code/.

Enlazamiento*

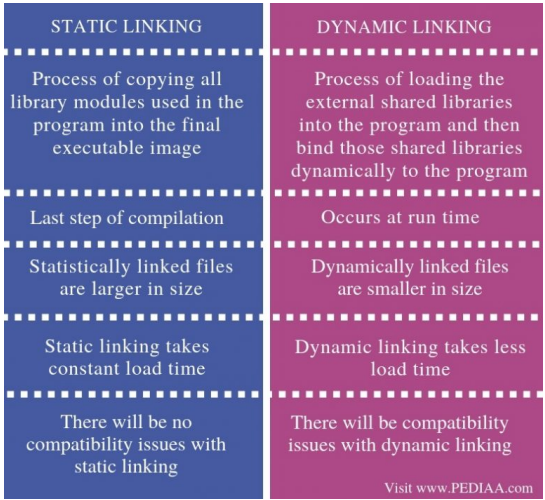


*Figura tomada de https://pediaa.com/what-is-the-difference-between-static-and-dynamic-linking/.

Enlazamiento

Ejemplo

- C++ usa enlazamiento dinámico por defecto, pero permite enlazamiento estático.
- Java usa solamente enlazamiento dinámico.

Enlazamiento

Ejemplo

(i) Enlazando estáticamente el programa `hw/hello-word.cc`:

```
$ cd hw
$ g++ -c hello-world.cc
$ g++ --static -o hw-static hello-world.o
```

(ii) Enlazando dinámicamente el programa `hw/hello-word.cc`:

```
$ g++ -o hw-dynamic hello-world.cc
```

(iii) Comparando los tamaños:

```
$ ls -lh hw-* | awk '{print $5, $9}'
17K hw-dynamic
2,4M hw-static
```

Recolección de basura

Características

- El recolector de basura (*garbage collector*) remueve automáticamente memoria que fue localizada dinámicamente por el programa pero que ya no es necesitada.
- Conflicto entre el control del programador y la memoria administrada automáticamente.
- El recolector de basura evita pérdidas de memoria.
- El recolector de basura impacta el rendimiento en tiempo de ejecución del programa.
- Lenguajes con recolector de basura requieren un sistema de tiempo de ejecución para ejecutar los programas.
- El recolector de basura se ejecuta en un hilo de ejecución.

Recolección de basura

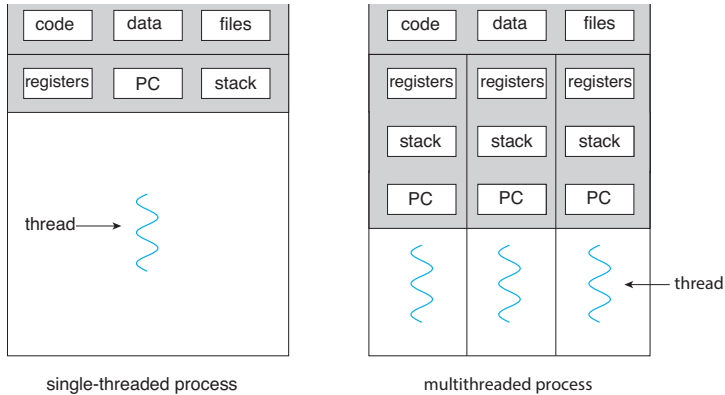
Ejemplo

- Java, Haskell y Python tienen recolector de basura.
- C y C++ no tienen recolector de basura.

Hilos de ejecución

Descripción

«*A **thread** is a basic unit of CPU utilization; it comprises a thread ID, a program counter (PC), a register set, and a stack. It shares with other threads belonging to the same process its code section, data section, and other operating-system resources, such as open files and signals.* [Silberschatz, Galvin y Gagne 2018, pág. 160 y Fig. 4.1]»



Hilos de ejecución

Ejemplo

A procesador de texto con tres hilos.*

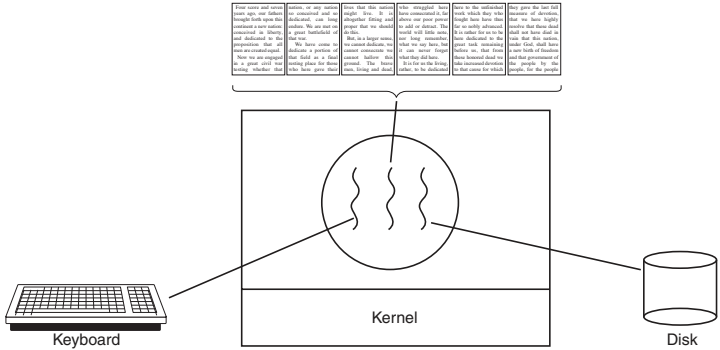


*Figura tomada de [Tanenbaum y Bos 2014, Fig. 2.7].

Apuntadores y referencias

Descripción

*«Los **apuntadores** (pointers) son la dirección de los datos en la memoria del computador. Los apuntadores se pueden utilizar en expresiones para crear nuevos apuntadores usando aritmética de apuntadores. En un lenguaje de programación un apuntador puede apuntar a cualquier lugar. Una **referencia** (reference) es más segura. Las referencias son algo así como apuntadores excepto que no se pueden utilizar en expresiones aritméticas. Ellas tampoco apuntan directamente a posiciones en la memoria. Cuando una referencia es desreferenciada, el sistema de tiempo de ejecución realiza la búsqueda en una tabla de referencias.» (pág. 130, traducción libre)*

(continua en la próxima diapositiva)

Apuntadores y referencias

Descripción (continuación)

«Esta diferencia entre referencias y apuntadores significa que podemos confiar con seguridad en que cada referencia apunte a un objeto real donde no necesariamente sabemos si un apuntador apunta a un espacio que podría liberarse de forma segura o no, ya que el apuntador podría ser el resultado de alguna aritmética de apuntadores. Las referencias son seguras para la recolección de basura. Los punteros no lo son.» (pág. 130, traducción libre)

Apuntadores y referencias

Paso de parámetros

El paso de parámetros a una función o a un método puede ser **por valor** (*call by value*) o **por referencia** (*call by reference*).

- Paso por valor: La función o el método recibe el valor de la variable que se pasa y crea una copia local de ésta. En consecuencia, no es posible modificar la variable pasada.
- Paso por referencia: La función o el método recibe la dirección de la variable que se pasa. En consecuencia, los cambios realizados por la función o el método se realizarán sobre la variable pasada.

Apuntadores y referencias

Paso de parámetros en C++

En C++ los parámetros pueden ser pasados por valor o por referencia.

Apuntadores y referencias

Paso de parámetros en C++

En C++ los parámetros pueden ser pasados por valor o por referencia.

Ejemplo

Mirar el archivo `oop/pointers-and-references.cc`.

Apuntadores y referencias

Ejercicio

¿Qué imprime el siguiente programa? ¿Por qué?

```
void f(int a, int& b) {
  a = a + b;
  b = a - b;
  a = a - b;
}

void g(int* a, int b) {
  *a = *a + b;
  b  = *a - b;
  *a = *a - b;
}
```

```
int main()
{
  int x = 1, y = 2;

  f(x,y);
  cout << "x:␣" << x;
  cout << "␣y:␣" << y << endl;

  g(&x,y);
  cout << "x:␣" << x;
  cout << "␣y:␣" << y << endl;
}
```

Apuntadores y referencias

Paso de parámetros en Java

En Java los parámetros siempre son pasados por valor, pero el lenguaje diferencia entre dos clases de parámetros: tipos de datos primitivos (**`boolean`**, **`char`**, **`byte`**, **`int`**, **`short`**, **`long`**, **`float`** y **`double`**) y referencias a objetos (*object references*).

Apuntadores y referencias

Paso de parámetros en Java

En Java los parámetros siempre son pasados por valor, pero el lenguaje diferencia entre dos clases de parámetros: tipos de datos primitivos (**`boolean`**, **`char`**, **`byte`**, **`int`**, **`short`**, **`long`**, **`float`** y **`double`**) y referencias a objetos (*object references*).

Ejemplo

Mirar el archivo `oop/MethodParameters.java`.

Interfaces e implementaciones

Descripción

*«Each class must have two parts: an interface and an implementation... The **interface** of a class is the one place where we assert all of the assumptions that a client may make about any instances of the class; the **implementation** encapsulates details about which no client may make assumptions.» (Booch, Maksimchuk, Engle, Young, Conallen y Houston 2007, pág. 51)*

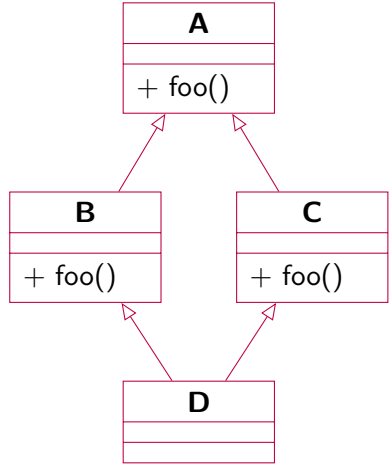
Interfaces e implementaciones

Ejemplo

- Las interfaces de Java son un conjunto de declaraciones de métodos sin implementación.
- C++ no tiene interfaces pero podemos pensar en los archivos de encabezado como en las «interfaces» de las clases.

Herencia múltiple

*«The **diamond problem** is an ambiguity that arises when two classes `B` and `C` inherit from `A`, and class `D` inherits from both `B` and `C`. If there is a method in `A` that `B` and `C` have overridden, and `D` does not override it, then which version of the method does `D` inherit: that of `B`, or that of `C`?» (Wikipedia 2023-09-05)*



Herencia múltiple

Ejemplo

- C++ suporta herencia múltiple.
  Mirar el archivo `oop/multiple-inheritance.cc`.
- Java no soporta herencia múltiple en las clases pero si lo hace en las interfaces.
  Mirar el directorio `oop/java-multiple-inheritance`.

Sobrecarga de funciones

Descripción

La **sobrecarga de funciones** es un característica que tienen algunos lenguajes de programación en la cual es posible definir varias funciones con el mismo nombre pero con diferentes parámetros.

Sobrecarga de funciones

Descripción

La **sobrecarga de funciones** es un característica que tienen algunos lenguajes de programación en la cual es posible definir varias funciones con el mismo nombre pero con diferentes parámetros.

Ejemplo (C++)

Mirar el archivo `oop/function-overloading.cc`.

Sobrecarga de funciones

Descripción

La **sobrecarga de funciones** es un característica que tienen algunos lenguajes de programación en la cual es posible definir varias funciones con el mismo nombre pero con diferentes parámetros.

Ejemplo (C++)

Mirar el archivo `oop/function-overloading.cc`.

Observación

Una buena práctica de programación consiste en que cuando una función es sobrecargada, cada una de las funciones debería implementar el mismo comportamiento.

Sobrecarga de funciones

Pregunta

Supongamos que hemos definido las siguientes funciones sobrecargadas en C++:

```
void fn(int, double);
void fn(double, int);
```

¿Qué sucede si llamamos la función `fn(0,0)`?

Manejo de excepciones

Descripción

*«**Exception handling** is the process of responding to the occurrence of exceptions—anomalous or exceptional conditions requiring special processing—during the execution of a program. In general, an exception breaks the normal flow of execution and executes a pre-registered **exception handler**.» (Wikipedia 2023-09-07)*

Manejo de excepciones

Soporte para el manejo de excepciones desde el lenguaje de programación

Algunos lenguajes de programación (*p. ej.* C++, Java, Python y Standard ML) tienen bloques de instrucciones *throw* y *try-catch* para el manejo de excepciones.

Manejo de excepciones

Soporte para el manejo de excepciones desde el lenguaje de programación

Algunos lenguajes de programación (*p. ej.* C++, Java, Python y Standard ML) tienen bloques de instrucciones *throw* y *try-catch* para el manejo de excepciones.

Ejemplo (C++)

Mirar el archivo `oop/exceptions.cc`.

Manejo de excepciones

Definición

*«A **signal** is an asynchronous notification sent to a process or to a specific thread within the same process in order to notify it of an event that occurred.» (Wikipedia 2019-10-02)*

Manejo de excepciones

Definición

> *«A **signal** is an asynchronous notification sent to a process or to a specific thread within the same process in order to notify it of an event that occurred.» (Wikipedia 2019-10-02)*

Ejemplo (C++)

Mirar el archivo `oop/signal-handling.cc`.

Referencias


Armstrong, Deborah J. (2006). The Quarks of Object-Oriented Development. Communications of the ACM 49.2, págs. 123-128. DOI: 10.1145/1113034.1113040 (vid. págs. 3, 4, 32-37).


Booch, Grady, Maksimchuk, Robert A., Engle, Michael W., Young, Bobbi J., Conallen, Jim y Houston, Kelli A. [1991] (2007). Object-Oriented Analysis and Design with Applications. 3.ª ed. Addison-Wesley (vid. págs. 32-37, 61).


Lee, Kent D. [2014] (2017). Foundations of Programming Languages. 2.ª ed. Undergraduate Topics in Computer Science. Springer (vid. pág. 2).


Silberschatz, Abraham, Galvin, Peter Baer y Gagne, Greg [2002] (2018). Operating System Concepts. 10.ª ed. Wiley (vid. pág. 51).


Tanenbaum, Andrew S. y Bos, Herbert [1992] (2014). Modern Operating Systems. 4.ª ed. Pearson (vid. pág. 52).