Física Computacional Avanzada

Introducción

Héctor Alvarez Pol

Departamento de Física de Partículas

Universidade de Santiago de Compostela

Física Computacional Avanzada - P1211101

Máster Universitario en Física - P1211V01

Curso 2020 / 2021

Presentación de la asignatura

P1211101 - Física computacional avanzada (Materia Obrigatoria) - Curso 2020/2021

Información | Profesores | Horarios

Información

- **Créditos ECTS**
- Créditos ECTS: 6.00
- **Total: 6.0**
- **Horas ECTS Criterios/Memorias**
- Clase Expositiva: 18.00
- Clase Interactiva Laboratorio: 24.00
- Horas de Titorías: 6.00
- Traballo do Alumno ECTS: 102.00
- **Total: 150.0**

Outros Datos

- **Tipo:** Materia Ordinaria Máster RD 1393/2007
- **Departamentos:** Física Aplicada, Física Aplicada, Física Aplicada, Electrónica e Computación, Electrónica e Computación, Departamento externo vinculado ás titulacións, Física de Partículas, Física de Partículas, Física de Partículas
- **Áreas:** Física Aplicada, Óptica, Electromagnetismo, Electrónica, Ciencia da Computación e Intelixencia Artificial, Área externa M.U en Física, Física da Materia Condensada, Física Atómica, Molecular e Nuclear, Física Teórica
- **Centro:** Facultade de Física
- **Convocatoria:** 1º Semestre de Titulacións de Grao/Máster
- **Docencia e Matrícula:** Primeiro Curso (1º 1ª vez)

Profesorado de la asignatura

P1211101 - Física computacional avanzada (Materia Obrigatoria) - Curso 2020/2021

Información | Profesores | Horarios

Profesores

Nome	Coordinador
ALVAREZ POL, HECTOR.	NON
HERNANDO MORATA, JOSE ANGEL.	NON
Martinez Hernandez, Diego.	NON
VAZQUEZ LOPEZ, RICARDO ANTONIO.	SI

Todos los profesores de esta asignatura somos físicos de distintas áreas de conocimiento, con experiencia profesional en distintas actividades de la Física Computacional.

Agenda y organización del tema de Simulación

Programa (fechas aproximadas, parón la primera semana de Noviembre):

• 22-23 de Septiembre: Introducción a la asignatura: UNIX, fundamentos de POO, ... (Héctor, 4 horas).

• 24 de Septiembre al 21 de Octubre: Introducción a Python, métodos numéricos (Diego, 30 horas).

• 22 de Octubre al 29 de Octubre: Métodos estadísticos avanzados (José Ángel, 10 horas).

• 30 de Octubre al 20 de Noviembre: Simulación. Números aleatorios, probabilidad, Monte-Carlo, problemas clásicos de simulación (Héctor, 16 horas).

• 24 de Noviembre al 14 de Diciembre: Cálculo simbólico (Ricardo, 30 horas).

Horario de clases Primeiro Cuadrimestre

	Luns	Martes	Mércores	Xoves	Venres
18:30-20:30		Grupo /CLE_01 3 (Informática) Horario detallado	Grupo /CLE_01 3 (Informática) Horario detallado	Grupo /CLE_01 3 (Informática) Horario detallado	Grupo /CLE_01 3 (Informática) Horario detallado

Calendario de exames - Curso

Centro	Data	Hora	Lugar	Convocatoria	Edición
Facultade de Física	21/01/2021	10:00	3 (Informática)	1º Cuadrimestre	1ª Oportunidade
Facultade de Física	21/06/2021	10:00	3 (Informática)	1º Cuadrimestre	2ª Oportunidade

Motivación de la asignatura

¿Por qué creemos necesaria la asignatura obligatoria de Física Computacional Avanzada?

Entendemos Física Computacional como un término amplio que abarca el conocimiento de **soluciones y herramientas computacionales** para el desarrollo de **métodos matemáticos, numéricos y simbólicos** relacionados con la resolución de problemas complejos en Física, así como la **modelación y simulación** de estos problemas.

Por tanto:

- Es un componente **esencial** en la formación científica y en particular en el campo de la Física.
- Corresponde a un **problema general e interdisciplinar** en Física, por tanto común a todos vosotros.
- Es una herramienta **potente para la resolución de problemas complejos**:
  - Permite el análisis y la categorización de situaciones físicas en sus componentes básicos.
  - Genera soluciones a problemas que no se pueden gestionar analíticamente o resultan demasiado complejos para tratarlos mediante otras aproximaciones.
  - Permite la visualización científica de las soluciones a los problemas propuestos.

Contenidos que comprende la asignatura

Secciones y contenidos básicos en cada sección:

• **Introducción:** Nociones básicas de **UNIX**. Introducción a los **lenguajes de programación**: lenguajes compilados e interpretados. Programación en **Python**. Técnicas avanzadas de programación: **Programación orientada a objetos y programación funcional**. Programación en **C++**.

• **Métodos numéricos:** Resolución de **EDO** y **EDP**. Métodos de **diferencias y elementos finitos**. **Métodos espectrales**.

• **Métodos de simulación:** **Problemas clásicos de simulación**: Modelo de Ising, percolación. Métodos de **Monte-Carlo**. Números **aleatorios y pseudoaleatorios**. Generación de **distribuciones de probabilidad**.

• **Métodos estadísticos avanzados:** Metodos **multivariables**. **Análisis de componentes principales**. Análisis de discriminantes. Análisis de Fischer. Análisis de Factores. **Redes neuronales**.

• **Cálculo simbólico**: Introducción. SimPy. Resolución de problemas de **álgebra lineal**. Resolución y representación de **ecuaciones diferenciales en derivadas parciales**. Resolución de **ecuaciones integro-diferenciales** directa y por el método de los momentos.

Objetivos que comprende la asignatura

Objetivos que se esperan alcanzar tras cursar esta asignatura:

El objetivo de esta materia es que el alumno adquiera las **competencias avanzadas en el campo de la física computacional** que no se adquieren en el grado y que le van a resultar **necesarias para resolver problemas complejos en distintas ramas de la física**, tanto teórica como experimental, como son:

- El **conocimiento** de los **sistemas operativos y los lenguajes y técnicas de programación** de uso común en física.
- La **destreza** para la **resolución mediante métodos numéricos de ecuaciones diferenciales e integrales, problemas algebraicos y problemas de minimización y optimización**.
- La **capacidad** de **diseñar modelos físicos** mediante simulación en el ordenador.
- La **competencia** en el **manejo de aplicaciones informáticas** para tratar problemas de física mediante el cálculo simbólico y el uso de técnicas matemáticas y gráficas avanzadas.

¡Conocimiento, destreza, capacidad y competencia que solo se consiguen mediante la práctica!
Es muy importante el **trabajo personal y la dedicación paciente al desarrollo** de vuestras habilidades computacionales.

Metodología

Metodología:

La materia tendrá un carácter fundamentalmente **práctico y aplicado**.

• Habrá un **número reducido de sesiones expositivas teóricas** para introducir los métodos. El resto del trabajo presencial serán **sesiones de prácticas con ordenador.**

• Se propondrán al alumno **trabajos de programación, cálculo y simulación** aplicados a distintos problemas de la física.

• Se propondrán **trabajos específicos que podrán estar relacionados con otras materias del máster** que el alumno esté cursando.

• El trabajo en las aulas se complementará con sesiones de tutoría y de trabajo personal del alumno.

Sistema de evaluación

Sistema de evaluación:

La evaluación de la materia consistirá básicamente en la **evaluación continua:**

• Es **obligatorio asistir a las clases expositivas e interactivas** y realizar las prácticas que se fijen en cada parte del curso.

• Se propondrán **trabajos o tareas específicos** donde el alumno pondrá en práctica los métodos y técnicas aprendidos a algún problema concreto de física, que puede estar relacionado con las otras materias del máster que el alumno esté cursando o tenga intención de cursar. Los trabajos correspondientes **se entregarán en el aula virtual una semana después de la última clase correspondiente a cada tema**. Se habilitará la correspondiente sección de tareas en el aula virtual con fecha estricta de entrega.

La actividad evaluable comprende:

• **Asistencia a las clases y realización de las prácticas. Examen final optativo. 60%.**

• **Presentación de trabajos o proyectos específicos 40%.**

Excepcionalmente, para **aquellos alumnos que no opten por la evaluación continua**, se podrá basar la nota en el examen final de la materia, siempre que el alumno realizara todas las prácticas propuestas durante las sesiones interactivas.

Bibliografía y recursos

Libros y recursos recomendados:

• M. Lutz, *Learning Python*, O'Reilly 2009.

• http://sympy.org/es/index.html

• B. Stroustrup: *El lenguaje de programación C++*, Addison-Wesley, 2009.

• S. Wolfram, *Mathematica : a system for doing mathematics by computer*, Addison-Wesley 1993.

• E. Weinstein: *Wolfram Mathworld*, http://mathworld.wolfram.com

• W.H. Press et al.: *Numerical recipes: the art of scientific computing*, Cambridge University Press, 2007.

• W. Cheney y D. Kincaid: *Numerical mathematics and computing*, T. Brooks/Cole, 2007

• D. W. Heermann, *Computer Simulation Methods in Theoretical Physics*, Springer 1990.

• T. Pang, *An introduction to computational physics*, Cambridge 2006.

• M.M. Woolfson, G.J. Pert, *An Introduction to Computer Simulation*, Oxford 1999.

• H. Gould, J. Tobochnik, W. Christian, *An introduction to computer simulation methods. Applications to physical systems*, Addison-Wesley.

• M.A. Kalos y P.A. Whitlock: *Monte Carlo methods*, Wiley, 2008

• I.T. Jolliffe *Principal Component Analysis*, second edition, Springer 2002.

• F. Husson, S. Le, J. Pages, *Exploratory Multivariate Analysis by Example Using R*, Chapman & Hall 2010.

• T. Hastie et al., *The elements of statistical learning*, Springer 2008.

¿Que necesitais para empezar?

Ahora necesitamos determinar donde empezamos el trabajo:

- Todos tenéis cuentas en el servidor UNIX de la facultad. ¿Sabéis acceder?
- ¿Cual es vuestro conocimiento de UNIX? ¿Os manejáis en un sistema de archivos?
- ¿Sabéis la diferencia entre programación estructurada y programación orientada a objetos?
- ¿Sabéis programar en algún lenguaje? ¿Cual es vuestra experiencia con la programación?
- ¿Sabéis Python? ¿Qué experiencia tenéis programando en lenguajes interpretados?
- ¿Y C ó C++?
- ¿Participais en algún trabajo de investigación? ¿Tenéis la intención de trabajar en algún grupo de la facultad? ¿Usaréis herramientas de simulación o análisis computacional avanzadas?

1. Introducción: UNIX

- ¿Qué es UNIX?
- Procesos, shell y sistema de archivos.
- Comandos básicos de UNIX.
- Aplicaciones importantes.

¿Que es UNIX?

UNIX designa el núcleo de un sistema operativo multiusuario y multitarea escrito en C.

En un sentido más amplio, comprende el **núcleo del sistema operativo** más un conjunto de **programas que permiten compilar** lenguajes de programación, **editar texto**, **interpretar comandos**, **manejar archivos y discos**, acceder a otras máquinas, **establecer comunicaciones**, enviar y recibir correo electrónico, manejar las colas de impresión...

Un sistema operativo (OS) gestiona y permite el acceso al hardware y el software de un ordenador de forma segura y eficiente. Controla todos los recursos de la computadora y proporciona la base sobre la que pueden escribirse los programas de aplicación.

Consta de los siguientes componentes jerarquicos:

- **Kernel**: parte del OS controla el hardware.
- **Librerias del sistema**: servicios básicos relacionados con el hardware, memoria, gestión de tareas, …
- **Shell/GUI**: acceso al usuario.
- Sistema de archivos.
- Utilidades del sistema y aplicaciones.

Características de un sistema UNIX

Características principales de UNIX y sistemas similares o basados en UNIX: Linux, MaOS…:

- **Modular**: **reutiliza su software**, con combinación de comandos simples en aplicaciones complejas.
- **Portable**: corre en un espectro de máquinas que van desde notebooks a supercomputadoras.
- **Flexible**: se adapta a muchas aplicaciones diferentes.
- **Potente**: dispone de muchos comandos y servicios ya incorporados.
- **Multiusuario**: atiende a muchas personas simultáneamente.
- **Multitarea**: puede gestionar varias tareas de forma simultánea. Manejo **dinámico** de **memoria**.
- **Elegante**: sus comandos son breves, coherentes, específicos para cada tarea y muy eficientes.
- Dispone de un estándar (**POSIX**) que debe cumplir todo sistema operativo que pretenda ser UNIX, lo que asegura una evolución predecible.
- Pensado para trabajar en **redes** de ordenadores (**protocolos de red incorporados**).
- Dispone de muchas **herramientas de software** (compiladores, editores, ...).

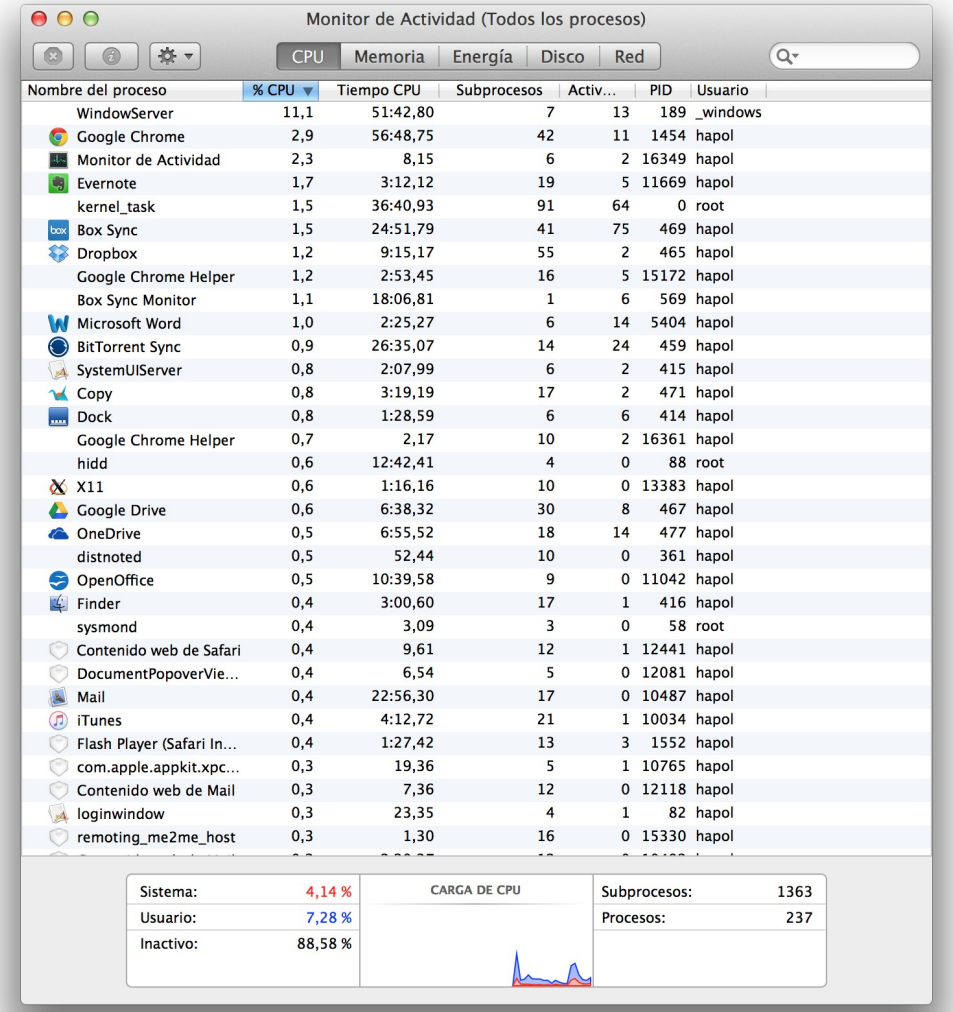
Componentes de un OS: procesos y shell

Procesos:

Un proceso es básicamente **un programa en ejecución**. Consta del programa ejecutable, datos, pilas, registros y toda la información necesaria para ejecutarse.

Un **intérprete de comandos (shell)** es un proceso que **lee los comandos de usuario** desde una terminal y **crea procesos hijo para ejecutar ese comando**. Se crea así un árbol de procesos en ejecución. Los procesos se manejan mediante **señales** que los obligan a suspender, reiniciar o terminar su acción. Las señales se utilizan también para comunicación entre procesos.

Cada usuario tiene asignado un **identificador de usuario (uid)**, y grupos de usuarios tienen un **identificador de grupo (gid)**. Un proceso tiene asignado el uid y gid del usuario que lo inició. Ambos identificadores se utilizan para proteger la información manejada por el proceso.



Componentes de un OS: sistema de archivos

Archivos y sistema de archivos.

Los **archivos contienen información** de forma permanente. La información se codifica en un conjunto de bits, almacenados en un dispositivo e identificado por un nombre y la descripción de la **carpeta o directorio** que lo contiene.

Esto crea una **jerarquía de directorios y archivos llamada sistema de archivos**. Un archivo se reconoce dando la ruta de acceso (path), que es la sucesión de directorios que permiten alcanzar el archivo a partir del directorio raíz (/).

En un sistema multiusuario, es preciso dar **privacidad** a los archivos de cada persona. UNIX utiliza un código de 9 bits en 3 grupos de 3 bits. Estos grupos correspondientes al **dueño**, el **grupo** y el **resto del mundo**.

Cada conjunto de 3 bits corresponde a permiso de **lectura**, **escritura** y **ejecución**.

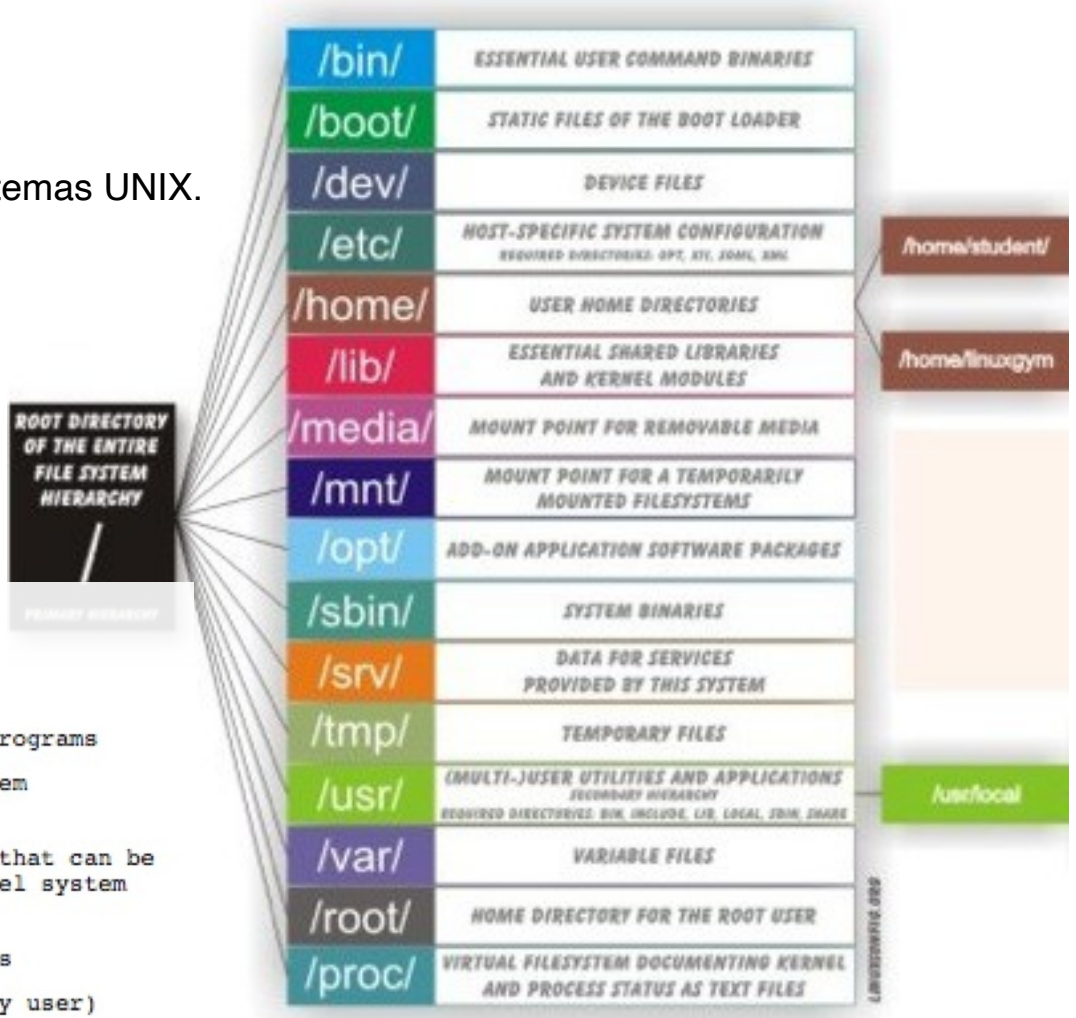
Se presentan como **rwx** (**Read**, **Write**, **eXecute**); cuando uno de estos permisos está denegado se sustituye la letra por - (como en r-- o en rw-).

```
MacBook-Pro-de-Hector:cal hapol$ ls -la
total 4472
drwxr-xr-x  22 hapol  staff      748 28 jul 23:52 .
drwxr-xr-x  38 hapol  staff     1292 28 jul 23:52 ..
-rw-r--r--   1 hapol  staff     1833 28 jul 23:52 CMakeLists.txt
-rw-r--r--   1 hapol  staff      705 28 jul 23:52 CaloLinkDef.h
-rw-r--r--   1 hapol  staff    47582 28 jul 23:52 R3BCalo.cxx
-rw-r--r--   1 hapol  staff     6833 28 jul 23:52 R3BCalo.h
-rw-r--r--   1 hapol  staff     2889 28 jul 23:52 R3BCaloContFact.cxx
-rw-r--r--   1 hapol  staff      435 28 jul 23:52 R3BCaloContFact.h
-rw-r--r--   1 hapol  staff    11286 28 jul 23:52 R3BCaloGeometry.cxx
-rw-r--r--   1 hapol  staff      551 28 jul 23:52 R3BCaloGeometry.h
-rw-r--r--   1 hapol  staff    80153 28 jul 23:52 R3BCaloHitFinder.cxx
-rw-r--r--   1 hapol  staff     6298 28 jul 23:52 R3BCaloHitFinder.h
-rw-r--r--   1 hapol  staff     2683 28 jul 23:52 R3BCaloHitFinderPar.cxx
-rw-r--r--   1 hapol  staff     2602 28 jul 23:52 R3BCaloHitFinderPar.h
-rw-r--r--   1 hapol  staff      892 28 jul 23:52 R3BGeoCalo.cxx
-rw-r--r--   1 hapol  staff      593 28 jul 23:52 R3BGeoCalo.h
-rw-r--r--   1 hapol  staff     1067 28 jul 23:52 R3BGeoCaloPar.cxx
-rw-r--r--   1 hapol  staff      810 28 jul 23:52 R3BGeoCaloPar.h
-rw-r--r--   1 hapol  staff  2078974 28 jul 23:52 calo.list
drwxr-xr-x  44 hapol  staff     1496 28 jul 23:52 perlScripts
-rw-r--r--   1 hapol  staff      275 28 jul 23:52 test.c
drwxr-xr-x  16 hapol  staff      544 28 jul 23:52 unpack
MacBook-Pro-de-Hector:cal hapol$ _
```

Componentes de un OS: sistema de archivos

Sistema de archivos.

Archivos presentes en la mayoría de los sistemas UNIX.



Directory	Typical Contents
/	The "root" directory
/bin	Essential low-level system utilities
/usr/bin	Higher-level system utilities and application programs
/sbin	Superuser system utilities (for performing system administration tasks)
/lib	Program libraries (collections of system calls that can be included in programs by a compiler) for low-level system utilities
/usr/lib	Program libraries for higher-level user programs
/tmp	Temporary file storage space (can be used by any user)
/home or /homes	User home directories containing personal file space for each user. Each directory is named after the login of the user.
/etc	UNIX system configuration and information files
/dev	Hardware devices
/proc	A pseudo-filesystem which is used as an interface to the kernel. Includes a sub-directory for each active program (or process).

Componentes de un OS: sistema de archivos

Algunos archivos esenciales presentes en todas las cuentas:

La shell utiliza algunos archivos que están presentes en vuestra cuenta:

- .profile
- .bashrc y/o .bash_profile
- .bash_logout
- .bash_history

Estos archivos pueden contener valores iniciales de las **variables de entorno** y **alias** de comandos. Son ejemplos de como se puede programar acciones en el sistema mediante **scripts (archivos de procesados por lotes)** del sistema o del shell. Los archivos script suelen ser identificados por el sistema a través de uno de los siguientes encabezamientos: **#!/bin/bash #!/bin/ksh #!/bin/csh**

Las **variables de entorno** son un conjunto de valores que pueden adoptar ciertas variables que el sistema utiliza para su funcionamiento, para definir como se comportan programas, para encontrar la localización de librarias necesarias para compilar o ejecutar aplicaciones, …

Las variables de entorno y los alias se definen mediante:

- **export variable_de_entorno = valor**
- **alias alias_del_comando = valor**

Componentes de un OS: shell / GUI

Interprete de comandos: la shell.

El intérprete de comandos de UNIX, o shell, es un proceso que muestra un indicador de comandos ($, %, # o algún otro identificador) y **aguarda que el usuario introduzca un comando**.

La interconexión de procesos se puede realizar a través de secuencias de comandos (**tubo** o pipeline), **indicado por el caracter** **|** (barra vertical). Un tubo es un seudoarchivo en el cual un primer proceso escribe su salida para ser leída por un segundo proceso; la salida del primero es la entrada del segundo. Se puede redireccionar la salida mediante un **redireccionamiento** con **>** o **>>**:

```
MacBook-Pro-de-Hector:cal hapol$ ls
CMakeLists.txt          R3BCaloContFact.cxx     R3BCaloHitFinder.cxx    R3BGeoCalo.cxx          calo.list
CaloLinkDef.h           R3BCaloContFact.h       R3BCaloHitFinder.h      R3BGeoCalo.h            perlScripts
R3BCalo.cxx             R3BCaloGeometry.cxx     R3BCaloHitFinderPar.cxx R3BGeoCaloPar.cxx       test.c
R3BCalo.h               R3BCaloGeometry.h       R3BCaloHitFinderPar.h   R3BGeoCaloPar.h         unpack
MacBook-Pro-de-Hector:cal hapol$ ls |wc
      20      20     311
MacBook-Pro-de-Hector:cal hapol$ ls |wc >> save.wc
```

Interface gráfica de usuario: X

En UNIX se separa la interface gráfica de usuario del sistema operativo. La GUI de UNIX es una capa separada que corre sobre el sistema operativo. La falta de una GUI no impide a UNIX ser funcional. **Hay un GUI accesible para todos los sistemas UNIX denominado X Window o simplemente X**.

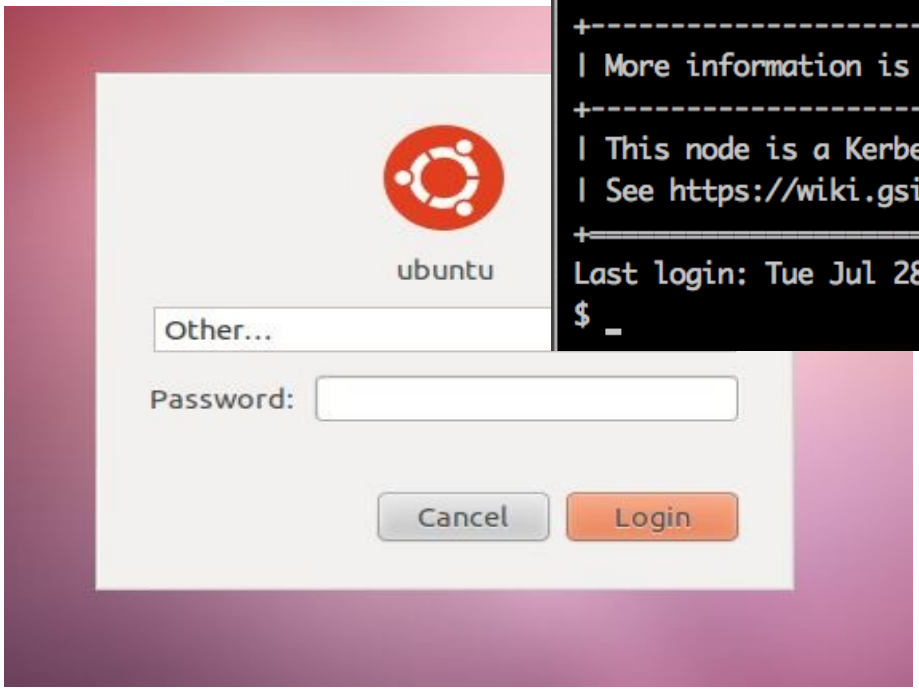
Acceso a un sistema UNIX

Acceso a un sistema UNIX:

Requiere estar registrado en el sistema y tener un espacio en el sistema de archivos con permisos de escritura. Cada usuario se identifica por un login (identificador) y un password (contraseña).

Conexión mediante ssh

```
MacBook-Pro-de-Hector:~ hapol$ ssh -X hapol@lxi001.gsi.de
hapol@lxi001.gsi.de's password:
Linux lxi079 3.2.0-4-amd64 #1 SMP Debian 3.2.68-1+deb7u2 x86_64
+=============================================================+
| Welcome to the Linux farm at GSI                            |
+-------------------------------------------------------------+
| If you have problems or other issues with the system        |
| please report them to linux-service@gsi.de                  |
+-------------------------------------------------------------+
| More information is available at https://wiki.gsi.de/Linux  |
+-------------------------------------------------------------+
| This node is a Kerberos client.                             |
| See https://wiki.gsi.de/Linux/Kerberos                      |
+=============================================================+
Last login: Tue Jul 28 12:10:16 2015 from 140.181.10.134
$ _
```



Comandos de UNIX: manual

man

UNIX dispone de un **manual técnico en línea** con información sobre comandos, archivos y otros elementos del sistema operativo. Para salir de la ayuda, presiona q (quit).

```
man(1)                                                          man(1)

NAME
       man - format and display the on-line manual pages

SYNOPSIS
       man  [-acdfFhkKtwW]  [--path]  [-m  system]  [-p  string]  [-C  config_file] [-M pathlist] [-P pager] [-B
       browser] [-H htmlpager] [-S section_list] [section] name ...

DESCRIPTION
       man formats and displays the on-line manual pages.  If you specify section, man only looks in  that  sec-
       tion  of the manual.  name is normally the name of the manual page, which is typically the name of a com-
       mand, function, or file.  However, if name contains a slash (/) then man interprets it as a file specifi-
       cation, so that you can do man ./foo.5 or even man /cd/foo/bar.1.gz.

       See below for a description of where man looks for the manual page files.

OPTIONS
       -C  config_file
              Specify the configuration file to use; the default is /private/etc/man.conf.  (See man.conf(5).)

       -M  path
              Specify  the  list  of directories to search for man pages.  Separate the directories with colons.
              An empty list is the same as not specifying -M at all.  See SEARCH PATH FOR MANUAL PAGES.

       -P  pager
              Specify which pager to use.  This option overrides the MANPAGER  environment  variable,  which  in
:_
```

Comandos de UNIX: gestión de archivos

Accediendo a la información en directorios y ficheros.

Cada usuario tiene un directorio propio, llamado home. Los comandos más importantes son:

• **pwd**: muestra el directorio actual.

• **cd (nombre_directorio)**: sin argumento, cambia hacia el directorio home. Si el argumento es un directorio, cambia a ese directorio, permitiendo el acceso a los fichero contenidos en él.

• **ls**: permite mostrar información sobre los archivos y directorios accesibles. Varias opciones permiten modificar su comportamiento para hacer su visualización mas completa o identificable.

• **mkdir nombre_directorio**: crea nuevos directorios.

• **rmdir nombre_directorio**: elimina directorios, siempre que estén vacios.

• **rm nombre_fichero**: elimina ficheros o directorios.

• **mv de_origen a_destino**: mueve ficheros o directorios de una dirección de origen a la de destino.

• **cp nombre_fichero nombre_copia_fichero**: copia ficheros.

• **ln -s nombre_fichero nuevo_enlace**: crea enlaces simbólicos a ficheros o directorios.

Comandos de UNIX: gestión de archivos

Mi directorio principal ó home

Desplazandome al directorio test

Permisos

Fecha de última modificación

```
MacBook-Pro-de-Hector:test hapol$ cd
MacBook-Pro-de-Hector:~ hapol$ pwd
/Users/hapol
MacBook-Pro-de-Hector:~ hapol$ ls
ACTAR           Copy            Dropbox         LISE            OneDrive        RPC             rootlogon.C
Applications    DOCENCIA        FRS             Library         PERSONAL        Sites           s438
Box Sync        Desktop         GENP            MACROS          Pictures        SolidWorks1.iso test
CODES           Documents       Google Drive    Movies          Public          VirtualBox VMs
CUNA            Downloads       IMPORTANTE      Music           R3B             anaconda
MacBook-Pro-de-Hector:~ hapol$ cd test
MacBook-Pro-de-Hector:test hapol$ ls
fichero.dat     textoPrueba.txt
MacBook-Pro-de-Hector:test hapol$ ls -l
total 16
-rw-r--r--  1 hapol  staff   5  4 sep 17:49 fichero.dat
-rw-r--r--  1 hapol  staff  27  4 sep 17:48 textoPrueba.txt
MacBook-Pro-de-Hector:test hapol$ ls -latr
total 16
drwxr-xr-x@ 107 hapol  staff  3638  4 sep 17:48 ..
-rw-r--r--    1 hapol  staff    27  4 sep 17:48 textoPrueba.txt
drwxr-xr-x    4 hapol  staff   136  4 sep 17:49 .
-rw-r--r--    1 hapol  staff     5  4 sep 17:49 fichero.dat
MacBook-Pro-de-Hector:test hapol$
```

Para comprobar las distintas opciones del comando ls y la descripción completa de la salida por pantalla en cada caso, introduce en la shell el comando:

man ls

Comandos de UNIX: gestión de archivos

Buscando información en directorios y ficheros.

Las herramientas de UNIX permiten gestionar, buscar, acceder e indexar la información en los ficheros de forma muy poderosa. La gestión completa de estas capacidades exige conocer fundamentos de “expresiones regulares” (regex ó regular expressions). De forma muy básica:

• **grep (egrep, fgrep, zgrep, zegrep, zfgrep)**: busca patrones dentro de ficheros.

• **find**: busca patrones dentro del sistema de archivos.

• **Expresiones regulares** es una secuencia de caracteres que forma un patrón de búsqueda, utilizada para la búsqueda de patrones de cadenas de caracteres u operaciones de sustituciones.

Ejemplos:

- egrep unix tmp.txt busca en el fichero tmp.txt las líneas que contienen la palabra unix.
- egrep '[Uu]nix' tmp.txt busca las líneas que contienen unix o Unix.
- egrep 'hel.' tmp.txt busca las líneas que contienen hel seguido de cualquier carácter.
- egrep 'ab*c' tmp.txt localiza las cadenas que empiecen por a, que continúen con 0 o más b, y que sigan con una c, por ejemplo: abbbc o aaacb, pero no axc o cba.
- egrep 't[^aeiouAEIOU][a-zA-Z]*' tmp.txt localiza las cadenas que empiecen por t, seguido de algún carácter no vocálico y 0 o más apariciones de otro carácter.

Comandos de UNIX: modificación de archivos

Modificando la información en directorios y ficheros.

- **chmod modo nombre_fichero**: cambia los permisos de acceso a ficheros o directorios.
- **chown dueño nombre_fichero**: cambia el dueño y el grupo de un fichero o directorio.
- **less nombre_fichero**: opuesto a more (humor fino).
- **emacs nombre_fichero**: completo editor de textos del proyecto GNU (ver hoja de referencia).
- **tar -cvf archivo_a_comprimir archivo_comprimido**: comprime y reune en un archivos simple grupos de archivos o directorios.
- **tar -xvf archivo_comprimido archivo_a_comprimir**: descomprime un fichero generado por tar.
- **gzip archivo_origen**: comprime ficheros.
- **gunzip archivo_origen**: descomprime (expande) ficheros.

Comandos de UNIX: conexiones seguras

UNIX permite acceder a otros ordenadores y copiar información de forma segura:

- **ssh -X usuario@maquina**: conectarse de forma segura a máquinas remotas.
- **scp fichero usuario@maquina:/localización**: copiar archivos de forma segura a maquinas remotas.
- **sftp usuario@maquina**: permite conexiones de ftp seguras.

```
MacBook-Pro-de-Hector:test hapol$ ls
fichero.dat     info.wc         textoPrueba.txt
MacBook-Pro-de-Hector:test hapol$ scp textoPrueba.txt hapol@lxi001.gsi.de:/u/hapol/.
hapol@lxi001.gsi.de's password:
textoPrueba.txt                                                100%   27     0.0KB/s   00:00
MacBook-Pro-de-Hector:test hapol$ ssh -X hapol@lxi001.gsi.de
hapol@lxi001.gsi.de's password:
Linux lxi078 3.2.0-4-amd64 #1 SMP Debian 3.2.68-1+deb7u1 x86_64
+-----------------------------------------------------------------+
| Welcome to the Linux farm at GSI                                |
+-----------------------------------------------------------------+
| If you have problems or other issues with the system            |
| please report them to linux-service@gsi.de                      |
+-----------------------------------------------------------------+
| More information is available at https://wiki.gsi.de/Linux      |
+-----------------------------------------------------------------+
| This node is a Kerberos client.                                 |
| See https://wiki.gsi.de/Linux/Kerberos                          |
+-----------------------------------------------------------------+
$ ls
2014_CALIFA_DATA  cros3_raw_data.C  ident_FRS.jpg   out.ps          R3BSimFile.root  testNov2010      ZvsAoQ_230.jpg
AIX               Desktop           mail           paraDavid       RAHADANAL        textoPrueba.txt
ALADINMAPS        geant4            Mail           params336nov01  RASNIK           unpacker
bin               go4_sis3220       main.pdf       params340nov01  run301_4334.lmd  web-bin
```

Comandos de UNIX: control de procesos

Finalmente veremos como podemos controlar el flujo de los procesos que gestiona un sistema UNIX:

Los programas pueden correr directamente bloqueando el acceso al usuario (se denomina en el **frente o foreground**) o pasando a un segundo plano, sin bloquear la introducción de nuevos comandos, aprovechando que el sistema es multitarea (en el **fondo o background**). Para cambiar:

- Añade **&** después de un comando para pasarlo al fondo.
- Un trabajo en el frente se puede suspender mediante **Ctrl-z** y pasarse al fondo mediante **Ctrl-c**.
- Para enviar un trabajo suspendido al fondo usa el comando **bg**.
- Para ver que comandos estan en ejecución usa **jobs**.
- Para pasar un comando al frente usa el comando **fg**.
- Para ver todos los comandos en ejecución ordenados usa **top**.
- Usa **kill numero_de_proceso** para matar un proceso (con precaución).

Ejercicios UNIX

[UNX1] Experimenta con los comandos:

echo hello word

date

hostname

who am i

who

ls

echo $SHELL

man ls

echo 5+4

echo 5+4 | bc -l

cal

cal 10 1492

time sleep 5

history

[UNX2] Intenta y describe el resultado de:

cd

pwd

ls -la

cd .

pwd

cd ..

pwd

ls -la

Ejercicios UNIX

[UNX3] Crea un directorio, añade o crea unos ficheros y cambia los permisos de un directorio y de los ficheros para permitir acceso a todo el mundo, acceso a tu grupo y solo acceso al usuario.

[UNX4] Prueba a comprimir un directorio y recuperarlo luego.

[UNX5] Corre el comando **sleep 15**, suspendelo, pasalo al fondo (bg) y comprueba su estado. Traelo de nuevo al frente (fg).

[UNX6] Prueba a combinar **ps -fae** y **grep** para mostrar el estado de los procesos que ejecutas.

[UNX7] Busca en el manual online de UNIX para que sirven y la sintaxis de los siguientes comandos: **apropos cat cd date exit echo finger head hostname id info less ls mail man mesg more passwd pwd rm talk tail touch whatis who who am i whoami**

1. Introducción: POO

- Justificación: La complejidad de los problemas a gestionar.
- Modelos de desarrollo de software.
- Definiciones: clases y objetos.
- Características esenciales de la POO.
- Relaciones entre clases y objetos.
- Otros ...

¿Un nuevo modelo de desarrollo de software?

¡¡¡El software es inherentemente complejo!!!

• El **problema a resolver puede resultar complejo** y el proceso de **desarrollo es difícil de gestionar**.

• Las aplicaciones informáticas son discretas. **Pequeños cambios pueden conducir a situaciones impredecibles**.

• **El hombre está seriamente limitado para tratar la complejidad**. Puede tener en mente 7±2 asuntos simultáneamente (con suerte y atención).

Atributos de la complejidad (Simon y Ando, Econometrica 29 (1961), 115):

• Los sistemas complejos comprenden subsistemas interrelacionados con algún tipo de relación jerárquica. Cada subsistema es en si mismo un sistema complejo, hasta llegar a componentes elementales. Los sistemas jerárquicos suelen componerse de unos pocos tipos de subsistemas que se unen y combinan de distintas formas.

• La elección de los componentes elementales o primitivos de un sistema es relativamente **arbitraria**, dependiendo del observador.

• Uniones o relaciones dentro de estos componentes son, generalmente, más fuertes que las relaciones entre distintos componentes (denominados de alta frecuencia).

• **Los sistemas complejos que funcionen deben evolucionar desde sistemas simples que funcionen**. Un sistema complejo no se crea a partir de uno con fallos, o de la nada.

Modelos de desarrollo de software

La solución para tratar con la complejidad es clásica: **Divide et impera**

• **Descomposición Algorítmica** (top-down o estructural): **rompe el sistema** en partes, cada una representando un pequeño paso del proceso. Métodos de diseño estructurados conducen a descomposiciones algorítmicas, donde la atención se centra en el flujo del sistema.

• **Descomposición Orientada a Objetos**: Trata de identificar **semánticamente** el dominio del problema. El entorno del problema se estudia como un conjunto de agentes autónomos (objetos) que colaboran para realizar un comportamiento complejo.

Algorítmica	**Orientada a objetos**
- Diagramas tipo arbol.	- Varias posibilidades.
- Desmenuza el problema.	- Identifica semánticamente el problema.
- Se programa en detalle.	- Se programa a lo grande.
- Lenguajes imperativos.	- Lenguajes declarativos.

Definiciones básicas

Análisis orientado a objetos es un método de análisis que examina los requerimientos desde la perspectiva de clases y objetos encontrada en el vocabulario original del problema.

Diseño orientado a objetos es un método de diseño que abarca el proceso de descomposición orientado a objetos y una notación para describir modelos lógicos y físicos, dinámicos y estáticos, del sistema bajo diseño.

Programación orientada a objetos es el método de implementación en el cual los programas se organizan como colecciones cooperantes de objetos, cada uno de los cuales representa un ejemplo de alguna clase, y cuyas clases son todas miembros de una jerarquía de clases unidas por relaciones (p.e herencia).

Un lenguaje es orientado a objetos si:

- **Soporta objetos que son abstracciones de datos**, mediatizados por operaciones, y con estados locales ocultos.
- Los objetos tienen un tipo asociado (**clase**).
- **Los tipos (clases) pueden heredar atributos** de supertipos (superclases).

¿Qué es un objeto?

• **Un objeto es una cosa tangible**, algo a que se puede aprehender intelectualmente o algo hacia lo que se puede dirigir una acción o pensamiento.

• Un objeto representa un item individual e identificable, o una entidad real o abstracta, con un papel definido en el dominio del problema.

• Un objeto tiene un **estado**, un **comportamiento** y una **entidad**.

• La estructura y el comportamiento de objetos similares se definen en sus clases comunes. El término objeto y ejemplo (instance) de una clase son intercambiables.

Identidad de un objeto

• **Identidad es la propiedad de un objeto que lo lleva a distinguirse de otros.**

¿Qué es un objeto?

Estado de un objeto

• El **estado de un objeto abarca todas las propiedades del objeto**, y los **valores actuales** de cada una de esas propiedades. Las propiedades de los objetos **suelen ser estáticas**, mientras los valores que toman estas propiedades cambian con el tiempo.

• El hecho de que los objetos tengan estado implica que ocupan un espacio, ya en el mundo físico, ya en la memoria del ordenador.

• No deben confundirse los objetos, que existen en el tiempo, son mutables, tienen estado, pueden ser creados, destruidos y compartidos..., con los valores (los asignados a una variable, por ejemplo) que son cantidades con las propiedades de ser atemporales, inmutables.

• **El estado de un objeto está influido por la historia del objeto**.

• El estado de un objeto representa el efecto acumulado de su comportamiento.

¿Qué es un objeto?

Comportamiento de un objeto

• **Comportamiento es como un objeto actua y reacciona**, en términos de sus cambios de estado y de los mensajes que intercambia.

• El comportamiento de un objeto representa su actividad externamente visible y testable. Son las **operaciones que una clase realiza** (llamadas también mensajes) las que dan cuenta de como se comporta la clase. Por operación se denota el servicio que una clase ofrece a sus clientes. Un objeto puede realizar cinco tipos de operaciones, con el proposito de provocar una reacción:

- **Modificador**: altera el estado de un objeto.
- **Selector**: accede al estado de un objeto, sin alterarlo.
- **Iterador**: permite a todas las partes de un objeto ser accedidas en un orden.
- **Constructor**: crea un objeto y/o inicializa su estado.
- **Destructor**: libera el estado de un objeto y/o destruye el objeto.

• C++ soporta, ademas de las operaciones, subprogramas libres. En la terminología de C++ las operaciones que un cliente puede realizar sobre un objeto se declaran como **funciones miembro**.

¿Qué es una clase?

Una clase es un conjunto de objetos que comparten una estructura y comportamiento comunes.

• Clase representa una abstracción, **la esencia que comparten los objetos**.

• **Un objeto es un ejemplo de una clase**.

• Un objeto no es una clase, y una clase no es un objeto (aunque puede serlo, p.e. en Smalltalk).

• Las clases actuan como intermediarias entre una abstracción y los clientes que pretenden utilizar la abstracción. De esta forma, la clase muestra:

  • **Visión externa de comportamiento (interface)**, que enfatiza la abstracción escondiendo su estructura y secretos de comportamiento.

  • **Visión interna (implementación)**, que abarca el código que se ofrece en la interface de la clase.

Características esenciales de la POO: abstracción

Abstracción

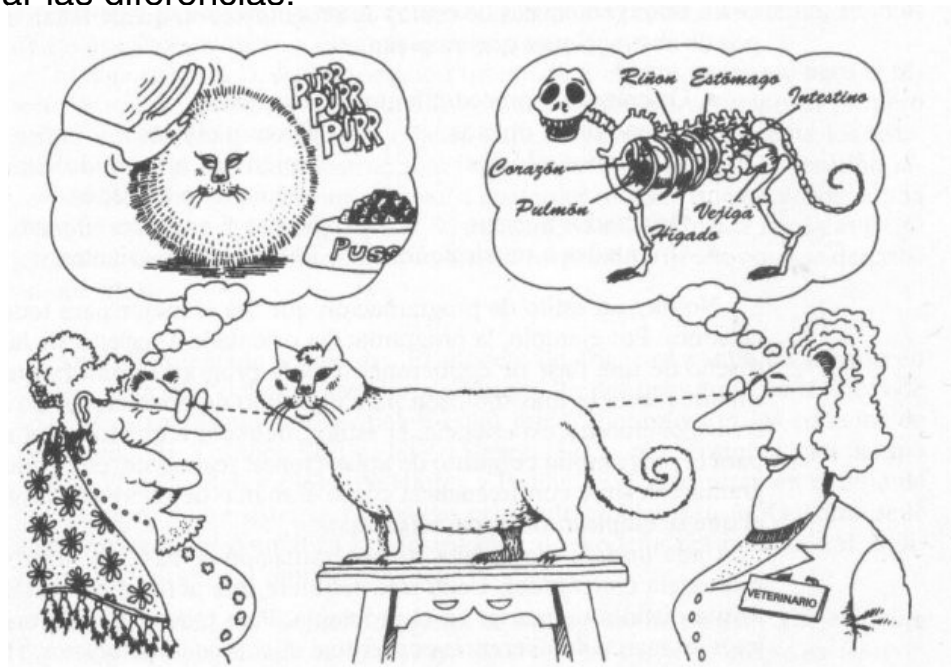
• **Diferencia y denota las características esenciales que distinguen a un objeto de otros tipos de objetos**, definiendo precisas fronteras conceptuales, relativas al observador.

• Surge del **reconocimiento de similaridades** entre ciertos objetos, situaciones o procesos en el mundo real.

• Decide concentrarse en estas similaridades e ignorar las diferencias.

• Enfatiza **detalles con significado para el usuario**, suprimiendo aquellos detalles que, por el momento, son irrelevantes o distraen de lo esencial.

• Deben seguir el "**principio de mínimo compromiso**", que significa que la interface de un objeto provee su comportamiento esencial, y nada más que eso. Pero también el "**principio de mínimo asombro**": capturar el comportamiento sin ofrecer sorpresas o efectos laterales.



Características esenciales de la POO: encapsulado

Encapsulado

• Es el proceso de **compartimentalización** de los elementos de una abstracción que constituyen su estructura y comportamiento. El encapsulado sirve para separar la interface de una abstracción y su implementación.

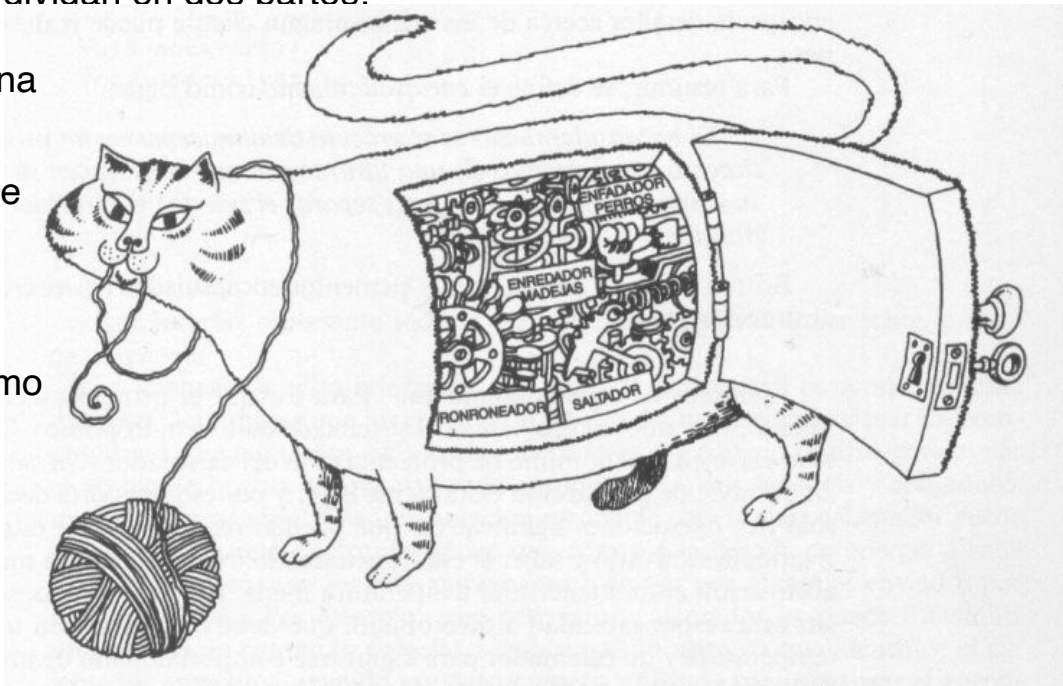
• Es un concepto complementario al de abstracción.

• El encapsulado **esconde** la implementación del objeto que no contribuye a sus características esenciales (principio de ocultación).

• El encapsulado da lugar a que las clases se dividan en dos partes:

- **Interface**: captura la visión externa de una clase, abarcando la abstracción del comportamiento común a los ejemplos de esa clase.
- **Implementación**: comprende la representación de la abstracción, así como los mecanismos que conducen al comportamiento deseado.

Se conoce también como ocultamiento o privacidad de la información.



Características esenciales de la POO: modularidad

Modularidad

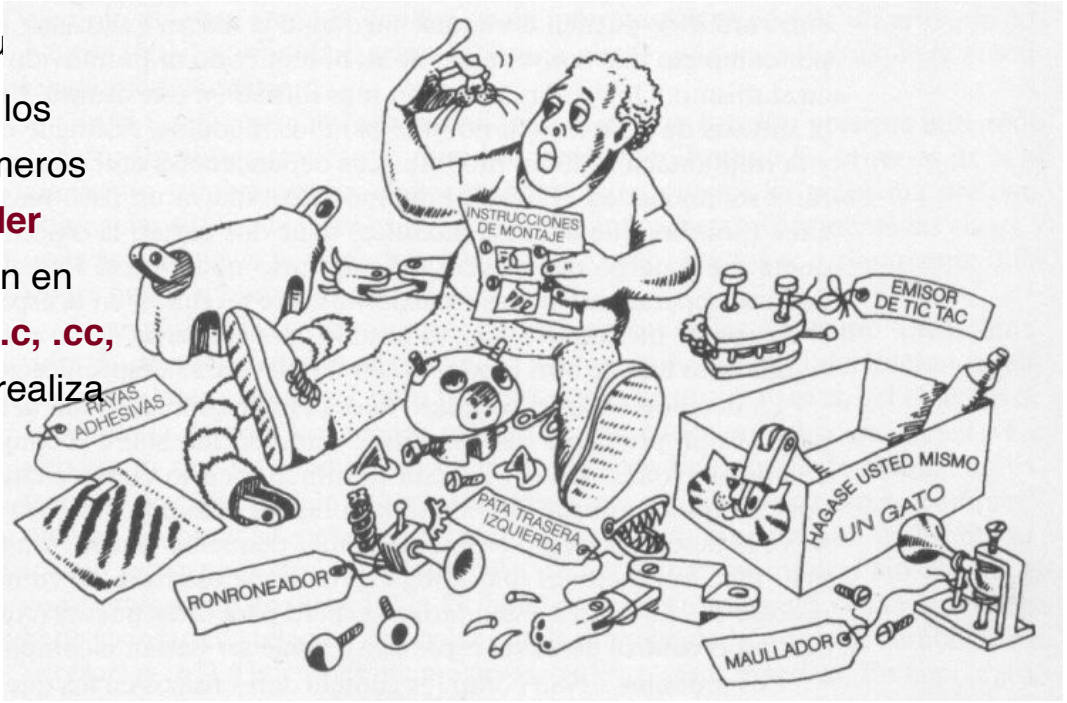
• Es la propiedad que tiene **un sistema que ha sido descompuesto** en un conjunto de módulos cohesivos y vagamente conexos.

• Cada módulo se puede compilar separadamente, aunque tengan conexiones con otros módulos.

• En un diseño estructural, modularización comprende el agrupamiento significativo de subprogramas. En **diseño orientado a objetos, la modularización debe ceñirse a la estructura lógica** elegida en el proceso de diseño.

• Dividir un programa en componentes individu

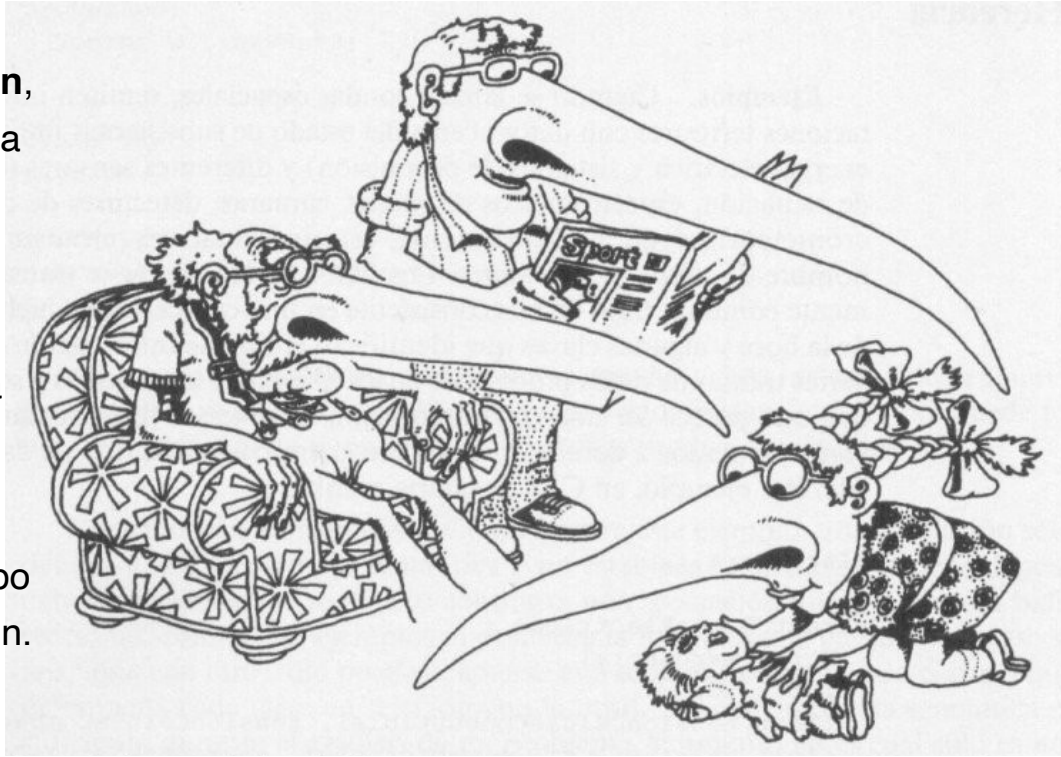
En C++: Se separan los módulos interface de los módulos con implementación, estando los primeros en **ficheros con extensión .h llamados header files**, mientras que los segundos se almacenan en **ficheros de implementación con extensión .c, .cc, .cp o .cpp**. La dependencia entre ficheros se realiza a través de la macro #include.



Características esenciales de la POO: jerarquización

Jerarquización

- Es una **clasificación** u **ordenación** de las abstracciones.
- **Por jerarquía denotamos el orden de relación** que se produce entre abstracciones diferentes.
- Los tipos de jerarquía más útiles:
  - **Herencia** (**generalización/especialización**, **padre/hijo**, jerarquía del tipo "**es un**"...): Una clase (subclase) comparte la estructura o comportamiento definido en otra clase, llamada superclase.
  - **Herencia múltiple**: Una clase comparte la estructura o comportamiento de varias superclases.
  - **Agregación**: Comprende relaciones del tipo "**es parte de**" al realizar una descomposición.



Características esenciales de la POO: tipificado

Tipificado (tipado)

• **Tipificar es la imposición de una clase a cada objeto**, de tal modo que objetos de diferentes tipos no se puedan intercambiar, o se puedan intercambiar solo de forma restringida.

• **Tipo** es una caracterización precisa de las propiedades estructurales y de comportamiento que comparten una coleción de entidades.

• Grosso modo, tipo y clase pueden considerarse sinónimos.

• Existen **lenguajes fuertemente tipificados** (Ada) y **débilmente tipificados** (C). Estos últimos soportan polimorfismo, mientras que los fuertemente tipificados no.

Características esenciales de la POO: otras...

Concurrencia

• Es la propiedad que **distingue un objeto activo de uno no activo**. Concurrencia permite que diferentes objetos actuen al mismo tiempo, usando distintos threads de control.

Persistencia

• Es la propiedad por la cual la **existencia de un objeto trasciende en el tiempo** (esto es, el objeto sigue existiendo despues de que su creador deja de existir) **o en el espacio** (esto es, la localización del objeto cambia respecto a la dirección en la que fue creado).

Polimorfismo

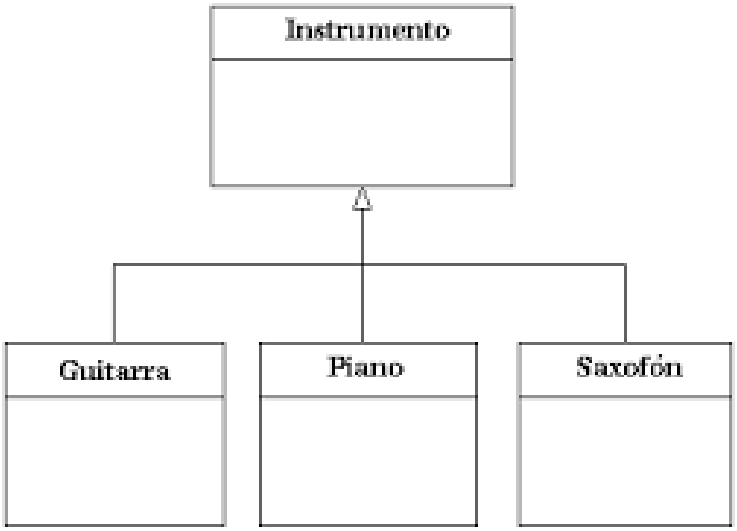
• Comportamientos diferentes, asociados a objetos distintos, pueden compartir el mismo nombre; se utilizará en cada caso el comportamiento correspondiente al objeto que se esté usando. Las referencias y las colecciones de objetos pueden contener objetos de diferentes tipos, y la invocación de un comportamiento en una referencia producirá el comportamiento correcto para el tipo real del objeto referenciado. Cuando esto ocurre en "tiempo de ejecución", esta última característica se llama asignación tardía o **asignación dinámica**. Algunos lenguajes proporcionan medios más estáticos (en "tiempo de compilación") de polimorfismo, tales como las plantillas y la sobrecarga de operadores de C++.

Relaciones entre clases

Representan tipos de compartición entre clases, o relaciones semánticas:

• **Asociación**. Indica relaciones de mandato bidireccionales (punteros ocultos en C++). Conlleva una dependencia semántica y no establece una dirección de dependencia. Tienen cardinalidad.

• **Herencia**. Por esta relación una clase (subclase) comparte la estructura y/o comportamiento definidos en una (herencia simple) o más (herencia múltiple) clases, llamadas superclases. Representa una relación del tipo "**es un**" entre clases. Una subclase aumenta o restringe el comportamiento o estructura de la superclase (o ambas cosas). Una clase de la que no existen ejemplos se denomina abstracta. C++ declara como virtuales todas aquellas funciones que quiere modificar en sus subclases.



Relaciones entre clases

Representan tipos de compartición entre clases, o relaciones semánticas:

• **Agregación**. Representa una relación del tipo "**tener un**" entre clases. Cuando la clase contenida no existe independientemente de la clase que la contiene se denomina agregación por valor y además implica contenido físico, mientras que si existe independientemente y se accede a ella indirectamente, es agregación por referencia.

• **Uso**. Es un refinamiento de la asociación donde se especifica cual es el cliente y cual el servidor de ciertos servicios, permitiendo a los clientes acceder sólo a las interfaces públicas de los servidores, ofreciendo mayor encapsulación de la información.

• **Ejemplificación**. Se usa en lenguajes que soportan genericidad (declaración de clases parametrizadas y argumentos tipo template). Representa las relaciones entre las clases parametrizadas, que admiten parámetros formales, y las clases obtenidas cuando se concretan estos parámetros formales, ejemplificados o inicializados con un ejemplo.

• **Metaclases**. Son clases cuyos ejemplos son a su vez clases. No se admiten en C++.

Relaciones entre clases y objetos

• **Todo objeto es el ejemplo de una clase**, y toda clase tiene 0 ó más objetos.

• Mientras **las clases son estáticas, con semántica, relaciones y existencia fijas** previamente a la ejecución de un programa, **los objetos se crean y destruyen rápidamente** durante la actividad de una aplicación.

• **El diseño de clases y objetos es un proceso incremental e iterativo**. Debe asegurar la optimización en los parámetros:

- **Acoplamiento**: Grado de acoplamiento entre módulos.
- **Cohesión**: Mide el grado de conectividad entre elementos de un módulo, y entre objetos de una clase.
- **Suficiencia**: Indica que las clases capturan suficientes características de la abstracción para conseguir un comportamiento e interacción eficiente y con sentido.
- **Completitud**: Indica que la interface de la clase captura todo el significado característico de una abstracción, escrito en el mínimo espacio.
- **Primitividad**: Las operaciones deben implementarse si dan acceso a una representación fundamental de la abstracción. Cuales son operaciones primitivas y cuales no (se pueden realizar a partir de otras) es un asunto subjetivo y afecta a la eficiencia en la implementación.

Para profundizar (notación, método, ...)

Para mas información, existe bibliografía específica, como:

- Booch, G., “Object-Oriented Analysis and Design with Applications”, 1st Edition, Addison-Wesley Professional (2011). (versión castellana en la biblioteca).

- Joyanes Aguilar, Luis, “Programación orientada a objetos”, Madrid : McGraw-Hill, D.L. (1996).

- Booch, G.; Rumbaugh, J.; Jacobson, I.: Unified Modeling Language User Guide, Addison-Wesley, 1998. ISBN: 0-201-57168-4. Existe traducción española: "El Lenguaje Unificado de Modelado", Addison-Wesley, Madrid, 1999, ISBN: 84-7829-028-1.

- Rumbaugh, J.; Jacobson, I.; Booch, G.: The Unified Modeling Language Reference Manual, Addison-Wesley, Reading, Mass., 1999. ISBN: 0-201-30998-X.

- Meyer, Bertrand. Object-Oriented Software Construction. Prentice Hall, segunda edición. Versión española: Construcción de software orientado a objetos, Prentice Hall Iberia, 1999.

- Stroustrup, B. The C++ Programming Language. Addison-Wesley Publishing Company, Reading, MA, edición especial, 1999. Existe edición española, Addison-Wesley, 2001

Las siguientes transparencias sirven para resumir algunos conceptos de análisis, diseño y modelización orientada a objetos.

Método orientado a objetos

Funciones del análisis y diseño:

- Satisface especificaciones funcionales.
- Determina las limitaciones del objetivo.
- Encuentra los requerimientos en la realización, y los recursos.
- Cumple (*hace cumplir*) criterios de diseño.
- Satisface restricciones, como tamaño o coste.
- Determina herramientas a utilizar.

Los distintos métodos de diseño comparten:

- **Notación**: un lenguaje para expresar el modelo.
- **Proceso**: actividades que conducen a la construcción ordenada del modelo.
- **Herramientas**: artefactos y reglas.

Método: la clasificación

Clasificar es la manera en la que ordenamos nuestro conocimiento. Clasificar es fundamentalmente un problema de agrupamiento. La identificación de clases y objetos es el asunto clave del diseño orientado a objetos. La identificación envuelve descubrimiento e invención. En Análisis orientado a objetos:

• Como primera aproximación clásica, **se proponen como clases y objetos los elementos de una categorización clásica**.

• Un análisis del comportamiento, de acuerdo con el agrupamiento conceptual, lleva a formar clases basadas en grupos de objetos que exhiben un comportamiento similar. Se agrupan por comunes responsabilidades (servicios ofrecidos en sus interfaces), formando jerarquías de especialización de responsabilidades.

• En el **análisis de dominio** se intentan identificar objetos, operaciones y relaciones que los expertos en el campo perciban como importantes en el dominio del problema.

Método: la clasificación

Clasificar es la manera en la que ordenamos nuestro conocimiento. En Análisis orientado a objetos:

• **Use-Case análisis o con guiones**: se usa conjuntamente con alguno de los anteriores. Se ejemplifican situaciones reales de utilización, a partir de unas condiciones iniciales. Debe provocar en los participantes una estructuración clara, y separación de responsabilidades.

• Las **tarjetas CRC** (Clase/Reponsabilidades/Colaboradores) sirven para analizar guiones. Se escriben en tarjetas los nombres de clases, sus reponsabilidades y sus colaboradores. Deben quedar claros los patrones jerárquicos entre clases, y el flujo de mensajes.

• En la **descripción informal** se escribe el problema en frases simples, subrayando nombres y verbos. Se convierte a los nombres en objetos y los verbos son operaciones sobre objetos. Obliga a trabajar sobre el vocabulario del problema, pero tiene el inconveniente de la imprecisión del lenguaje, a la hora de describir comportamientos precisos.

• El análisis de estructuras usa métodos de programación estructural, de los que deriva los elementos del modelo de objetos.

Método orientado a objetos

Abstracciones llave

• Son **clases u objetos que forman parte del vocabulario del problema**. Identifican aquellas abstracciones que acotan y aclaran el dominio de trabajo.

• Su identificación requiere descubrimiento e invención. Esta identificación inicial da lugar a abstracciones que, en muchos casos, resultan erróneas.

• Requieren un proceso de refinado y estructuración jerárquica. Se aconseja seguir las reglas de notación:

- Los **objetos deben denominarse con nombres propios y las clases tienen nombres comunes**.
- Operaciones **modificadoras deben denominarse con verbos activos, mientras las operaciones de selección usan preguntas o nombres con el verbo ser**.

Método orientado a objetos

Mecanismos

• Denotan decisiones estratégicas que consideran las actividades de colaboración de diferentes tipos de objetos.

• Representan patrones de comportamiento, sobre los que se decide cómo cooperan las colecciones de objetos.

• Los mecanismos permiten considerar el trabajo conjunto de las clases para producir un comportamiento complejo.

Método orientado a objetos: notación

• **La notación debe ser un vehículo para capturar la forma de funcionar, el comportamiento y la estructura de un sistema**.

• No será posible encontrar la información relevante en un único diagrama.

• Debe existir un equilibrio entre la complejidad de la notación y la precisión requerida al hacer el modelo.

• La notación no es un fin en si misma.

• Estas herramientas **no garantizan un diseño correcto, pero ayudan al diseñador a estructurar su conocimiento**. El programador puede centrarse en problemas de fondo, olvidándose de la forma.

• Para comprender un sistema orientado a objetos es necesario observar las clases y objetos de acuerdo con los modelos:

- **Físico/lógico**: La visión lógica describe la existencia y significado de las abstracciones llave y de los mecanismos del espacio del problema, o que definen la arquitectura del sistema. La visión física describe la composición del software y hardware concreta de la implementación o contexto del sistema. En esta descomposición se usan los diagramas de clase, de objeto, de modulo y de proceso.
- **Estático/dinámico**: los diagramas basados en los anteriores modelos son estáticos, pero los objetos se crean y destruyen, envian mensajes y dependen de triggers. El comportamiento dinámico se estudia los diagramas de transición de estado y de interacción.

Notación: UML

Los 14 tipos de diagramas de UML 2.2, divididos en 7 pertenecientes a información estructural y otros 7 a información de comportamiento. El diagrama inferior representa su organización jerarquica.