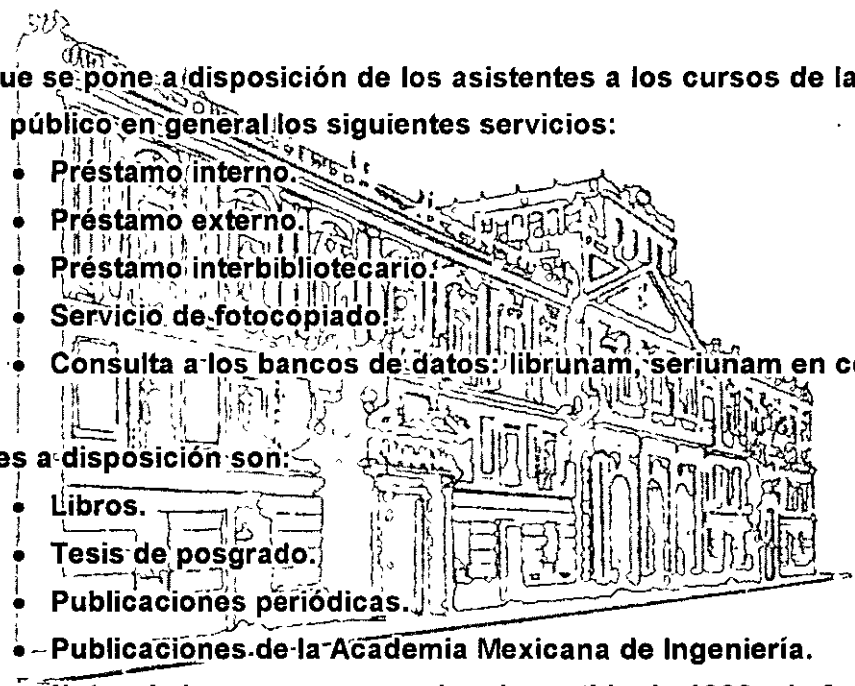
FACULTAD DE INGENIERIA U.N.A.M.
DIVISION DE EDUCACION CONTINUA

CENTRO DE INFORMACIÓN Y DOCUMENTACIÓN
" ING. BRUNO MASCANZONI "

El Centro de Información y Documentación Ing. Bruno Mascanzoni tiene por objetivo satisfacer las necesidades de actualización y proporcionar una adecuada información que permita a los ingenieros, profesores y alumnos estar al tanto del estado actual del conocimiento sobre temas específicos, enfatizando las investigaciones de vanguardia de los campos de la ingeniería, tanto nacionales como extranjeras.



Es por ello que se pone a disposición de los asistentes a los cursos de la DECFI, así como del público en general los siguientes servicios:

- **Préstamo interno.**
- **Préstamo externo.**
- **Préstamo interbibliotecario.**
- **Servicio de fotocopiado.**
- **Consulta a los bancos de datos: librunam, seriunam en cd-rom.**

Los materiales a disposición son:

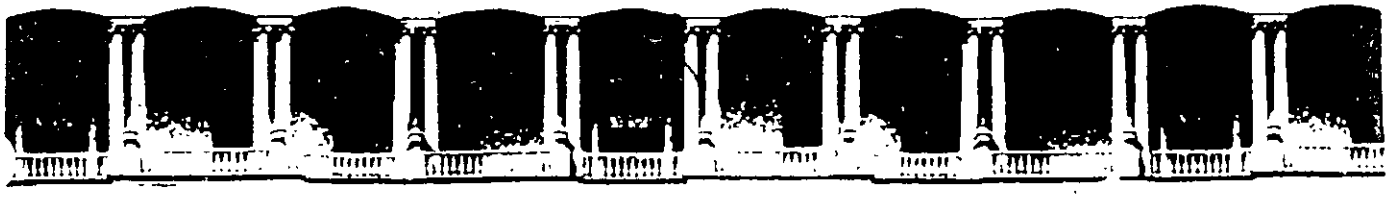
- **Libros.**
- **Tesis de posgrado.**
- **Publicaciones periódicas.**
- **Publicaciones de la Academia Mexicana de Ingeniería.**
- **Notas de los cursos que se han impartido de 1988 a la fecha.**

En las áreas de ingeniería industrial, civil, electrónica, ciencias de la tierra, computación y, mecánica y eléctrica.

El CID se encuentra ubicado en el mezzanine del Palacio de Minería, lado oriente.

El horario de servicio es de 10:00 a 14:30 y 16:00 a 17:30 de lunes a viernes.

Palacio de Minería Calle de Tacuba 5 Primer piso Deleg. Cuauhtémoc 06000 México, D.F. APDO. Postal M-2285
Teléfonos: 512-8955 512-5121 521-7335 521-1987 Fax 510-0573 521-4020 AL 26



FACULTAD DE INGENIERIA U.N.A.M.
DIVISION DE EDUCACION CONTINUA

A LOS ASISTENTES A LOS CURSOS

Las autoridades de la Facultad de Ingeniería, por conducto del jefe de la División de Educación Continua, otorgan una constancia de asistencia a quienes cumplan con los requisitos establecidos para cada curso.

El control de asistencia se llevará a cabo a través de la persona que le entregó las notas. Las inasistencias serán computadas por las autoridades de la División, con el fin de entregarle constancia solamente a los alumnos que tengan un mínimo de 80% de asistencias.

Pedimos a los asistentes recoger su constancia el día de la clausura. Estas se retendrán por el periodo de un año, pasado este tiempo la DECFI no se hará responsable de este documento.

Se recomienda a los asistentes participar activamente con sus ideas y experiencias, pues los cursos que ofrece la División están planeados para que los profesores expongan una tesis, pero sobre todo, para que coordinen las opiniones de todos los interesados, constituyendo verdaderos seminarios.

Es muy importante que todos los asistentes llenen y entreguen su hoja de inscripción al inicio del curso, información que servirá para integrar un directorio de asistentes, que se entregará oportunamente.

Con el objeto de mejorar los servicios que la División de Educación Continua ofrece, al final del curso deberán entregar la evaluación a través de un cuestionario diseñado para emitir juicios anónimos.

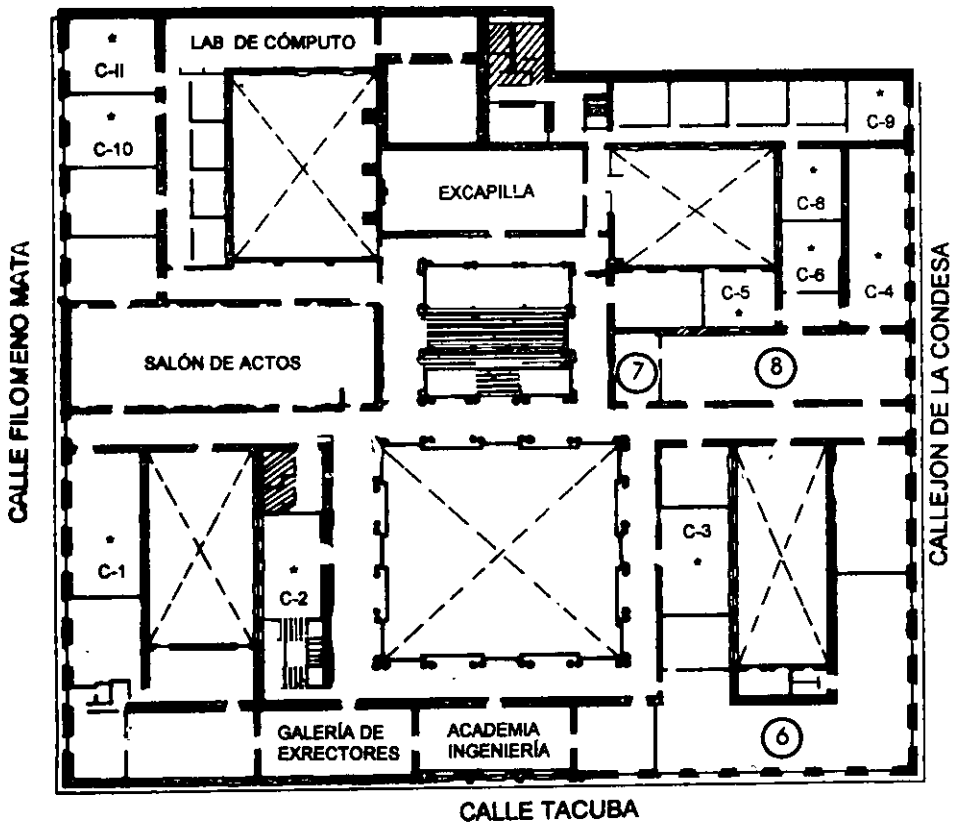
Se recomienda llenar dicha evaluación conforme los profesores impartan sus clases, a efecto de no llenar en la última sesión las evaluaciones y con esto sean más fehacientes sus apreciaciones.

Atentamente

División de Educación Continua.

Palacio de Minería Calle de Tacuba 5 Primer piso Deleg. Cuauhtémoc 06000 México, D.F APDO. Postal M-2285
Teléfonos. 5512-8955 5512-5121 5521-7335 5521-1987 Fax 5510-0573 5521-4021 AL 25

PALACIO DE MINERIA



1er. PISO

GUÍA DE LOCALIZACIÓN

1. ACCESO
2. BIBLIOTECA HISTÓRICA
3. LIBRERÍA UNAM
4. CENTRO DE INFORMACIÓN Y DOCUMENTACIÓN "ING. BRUNO MASCANZONI"
5. PROGRAMA DE APOYO A LA TITULACIÓN
6. OFICINAS GENERALES
7. ENTREGA DE MATERIAL Y CONTROL DE ASISTENCIA
8. SALA DE DESCANSO

SANITARIOS

* AULAS

DIVISIÓN DE EDUCACIÓN CONTINUA
FACULTAD DE INGENIERÍA U.N.A.M.
CURSOS ABIERTOS

DIVISIÓN DE EDUCACIÓN CONTINUA
DEC

DIVISION DE EDUCACION CONTINUA
FACULTAD DE INGENIERIA, UNAM
CURSOS ABIERTOS

DIVISIÓN DE EDUCACIÓN CONTINUA

CURSO: ***CC03 Introducción a la Programación***
FECHA: ***4 al 7 de febrero del 2002***

EVALUACIÓN DEL PERSONAL DOCENTE

(ESCALA DE EVALUACIÓN: 1 A 10)

CONFERENCISTA	DOMINIO DEL TEMA	USO DE AYUDAS AUDIOVISUALES	COMUNICACIÓN CON EL ASISTENTE	PUNTUALIDAD
TEC. YAZMIN VEGA HERNANDEZ				

Promedio _____

EVALUACIÓN DE LA ENSEÑANZA

CONCEPTO	CALIF.
ORGANIZACIÓN Y DESARROLLO DEL CURSO	
GRADO DE PROFUNDIDAD DEL CURSO	
ACTUALIZACIÓN DEL CURSO	
APLICACIÓN PRACTICA DEL CURSO	

Promedio _____

EVALUACIÓN DEL CURSO

CONCEPTO	CALIF.
CUMPLIMIENTO DE LOS OBJETIVOS DEL CURSO	
CONTINUIDAD EN LOS TEMAS	
CALIDAD DEL MATERIAL DIDÁCTICO UTILIZADO	

Promedio _____

Evaluación total del curso _____

Continúa...2

1. ¿Le agradó su estancia en la División de Educación Continua?

SI ☐ NO ☐

Si indica que "NO" diga porqué: _____

_____

2. Medio a través del cual se enteró del curso

Periódico *La Jornada*	
Folleto anual	
Folleto del curso	
Gaceta UNAM	
Revistas técnicas	
Otro medio (Indique cuál)	

3. ¿Qué cambios sugeriría al curso para mejorarlo?

_____

_____

_____

_____

_____

_____

4. ¿Recomendaría el curso a otra(s) persona(s) ?

SI ☐ NO ☐

5. ¿Qué cursos sugiere que imparta la División de Educación Continua?

_____

_____

_____

_____

_____

_____

6. Otras sugerencias:

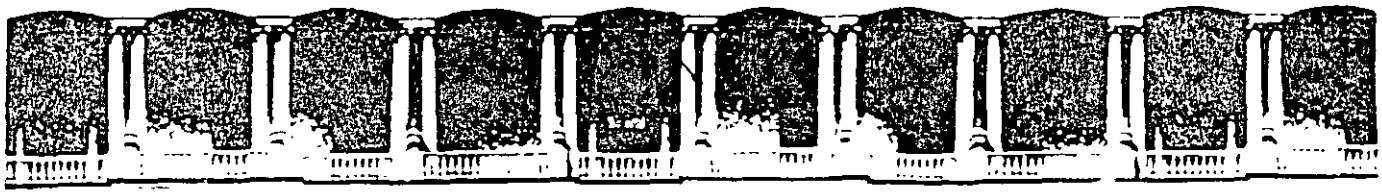
_____

_____

_____

_____

_____



FACULTAD DE INGENIERIA U.N.A.M.
DIVISION DE EDUCACION CONTINUA

INTRODUCCION A LA PROGRAMACION

F E B R E R O 2 0 0 2

Palacio de Mineria Calle de Tacuba 5 Primer piso Deleg Cuauhtémoc 06000 Mexico, D F APDO. Postal M-2285
Teléfonos 5512-8955 5512-5121 5521-7335 5521-1987 Fax 5510-0573 5521-4021 AL 25

I. Que es la Programación.

La programación es una de las principales herramientas que existen en la actualidad para la solución de problemas en todas las áreas de la ocupación humana, pero para poder responder de manera mas clara la pregunta que da nombre a este capitulo es necesario establecer algunas definiciones básicas involucradas con la programación.

1. Algoritmo.

Un algoritmo se define como la secuencia de pasos o instrucciones para resolver un problema dado. Cuando se involucran procesos matemáticos, se trata de algoritmos numéricos; cuando se habla de procesos en general hablamos de algoritmos no numéricos.

2. Lenguaje de Programación.

Existen dos tipos de lenguajes de programación:

- *De bajo nivel.*

  Consiste de instrucciones dadas directamente en el lenguaje de la máquina en código binario, que son introducidas a la máquina mediante lenguajes de mnemónicos conocidos genéricamente como ensambladores

- *De alto nivel.*

  En estos lenguajes se utilizan instrucciones que no pertenecen a la máquina si no al lenguaje especifico que se

esta utilizando, por lo regular estas instrucciones se encuentran en ingles.

3. Instrucciones.

Una instrucción es una acción que se ejecuta dentro de un programa esta instrucción puede o no utilizar o modificar datos del programa.

4. Los Datos.

Los datos son toda aquella información que es utilizada o modificada en la ejecución de un programa

5. Los Compiladores.

En realidad las computadoras tienen sólo un lenguaje de programación: el lenguaje de máquina. Cuando se programa en un lenguaje de alto nivel, existe además un programa llamado **compilador** que lo traduce, a su vez, en el lenguaje de máquina para su interpretación por la computadora. En este caso las instrucciones escritas en el lenguaje de alto nivel se conocen como programa fuente y las instrucciones del programa en lenguaje de máquina se conocen como programa objeto. La ventaja de utilizar programas compilados radica en que es mas rápida su fabricación, además de que pueden ser almacenados para su posterior ejecución en lenguaje máquina.

Establecidos estos conceptos podemos definir a la programación como:

"La solución de problemas mediante uno o mas algoritmos, implementados mediante algún conjunto de instrucciones especificas"

Una vez lograda esta definición útil para los fines de este curso podemos adentrarnos en los diferentes estilos de programación.

II. Programación Estructurada

1. Estructuras de Control.

IF

Esta estructura de control se emplea para elegir entre dos diferentes cursos de acción.

Sintaxis.

La sintaxis de esta estructura es la siguiente:

```
if( condicion )
{
    Sentencias a ejecutar
    si la condición es verdadera.

}
```

IF/ELSE.

Esta estructura de control amplia las capacidades de la estructura IF, pues nos permite seleccionar entre mas de dos posibles cursos de acción.

Sintaxis.

```
if( condicion )
{
```

Sentencias a ejecutar
si la condición es verdadera

}
else if(*segunda_condicion* **)**
{
Sentencias a ejecutar
si la segunda_condicion
es verdadera
}
else if(..... **)**
{
.....
}
.....

WHILE.

Esta estructura nos permite repetir una acción mientras que cierta condición se mantenga verdadera.

Sintaxis.

while(*condición* **)**
{
Sentencias a repetir
Mientras la condición
sea verdadera
}

DO/WHILE.

Su función es la misma que la de la estructura **WHILE**, solo que **DO/WHILE** nos garantiza que el proceso se realizara por lo menos una vez.

Sintaxis.

do
{
Sentencias que se ejecutaran ,
y que podrán repetirse mientras
la condición sea verdadera

}while(*condición* **);**

FOR.

Esta estructura nos permite realizar una acción un número determinado de veces.

Sintaxis.

for(*contador ; condición del contador ; incremento al contador***)**
{
Sentencias a ejecutar
hasta que se alcance la
condición del contador
}

SELECT/CASE.

Esta estructura nos permitirá realizar selecciones mas complejas que las realizadas con **IF** o **IF/ELSE**.

Sintaxis.

Select(seleccion **)**
{

case A:

Sentencias a ejecutar
si la selección es igual a A

case B:

Sentencias a ejecutar
si la selección es igual a B

default:

Sentencias a ejecutar si
no se encuentra el valor
de la selección.

}

2. Funciones.

La mejor manera para desarrollar un programa grande es dividiéndolo en pequeños módulos que realicen tareas especificas, estos módulos son conocidos como funciones o procedimientos. Al diseñar nuestros programas con funciones conseguiremos facilitar su diseño, implementación, operación y mantenimiento.

Definición de funciones.

Las funciones suelen estructurarse con los elementos listados a continuación:

```
tipo_retorno nom_Funcion( parametro1, paramtro2,... )
 {
    Sentencias por realizar en la función
    return valor de regreso;
 }
```

tipo_retorno: Con esta parte de la declaración especificamos el tipo de valor que regresara la función. Para indicar que la función no regresara un valor se utiliza la palabra reservada **void**.

nom_Funcion: El nombre de la función es con el que podremos llamarla a lo largo de nuestro programa.

parametro1, paramtro2,...: La lista de parámetros representa el tipo, nombre y orden con que serán enviados los parámetros a la función, en el momento en que sea invocada.

return: Con esta sentencia se regresa el control a quien haya invocado a la función, así como el valor de retorno de la función.

III. *Programación orientada a objetos*

3.1 Conceptos Básicos.

3.1 Mecanismos básicos.

Los mecanismos básicos de la orientación a objetos son los objetos, mensajes y los métodos, clases y variables instancia (o modelo) y herencia. Todos los sistemas que merecen la descripción de orientado a objetos contienen estos mecanismos esenciales, aunque los mecanismos pueden no estar realizados (o denominados) exactamente de la misma forma.

3.1.1 Objetos.

Se definirá un objeto como un concepto, abstracción o cosa con limites bien definidos y con significado a efectos del problema que se tenga entre manos. Los objetos tienen dos propósitos: promover la comprensión del mundo real y proporcionar una base practica para la implementación por computadora. La descomposición de un problema en objetos depende del juicio y de la naturaleza del problema. No existe una única representación correcta.

Un programa tradicional consta de procedimientos y datos. Un programa orientado a objetos consta solamente de objetos que contienen tanto los procedimientos como los datos. Por decirlo de otra forma, los objetos son módulos que contienen los datos y las instrucciones que operan sobre esos datos. Así, dentro de los objetos residen los datos de los lenguajes convencionales, como por ejemplo, números, matrices (arrays o arreglos), cadenas de caracteres y registros, así como cualquier función, instrucción o subrutina que opere sobre ellos. Los objetos, por tanto, son entidades que tienen atributos (datos) y formas de comportamiento (procedimientos) particulares.

Los objetos llevan los nombres de los elementos de interés desde el dominio de la aplicación. Por ejemplo, en una aplicación de procesamiento de textos,

es probable que un objeto sea llamado párrafo. En una aplicación de contabilidad, la hoja balance de junio es un objeto probable. Desde la perspectiva del usuario, los objetos proporcionan el comportamiento deseado. Un párrafo puede aceptar revisión y realinear sus márgenes. La hoja balance del mes de junio puede imprimirse o consolidarse con las de otros meses para constituir un informe cuatrimestral. Desde la perspectiva del programador, los objetos son módulos de una aplicación que funcionan juntos para proporcionar una funcionalidad general.

Es importante señalar que todos los objetos poseen su propia identidad y se pueden distinguir entre si. Dos manzanas del mismo color, forma y textura siguen siendo manzanas individuales. El termino identidad significa que los objetos se distinguen por su existencia inherente y no por las propiedades descriptivas que puedan tener.

Las aplicaciones pueden constar de diferentes clases de objetos. Un objeto activo es aquel que se comprende su propio hilo de control, mientras que un objeto pasivo no. Los objetos activos suelen ser autónomos lo que quiere decir que pueden exhibir algún comportamiento sin que ningún otro objeto opere sobre ellos. Los objetos pasivos, por otra parte, solo pueden padecer un cambio de estado cuando se actúa explícitamente sobre ellos. De este modo los objetos activos de un sistema sirven como raíces de control.

3.1.2 Mensajes y Métodos.

En la mayoría de los lenguajes de programación orientados a objetos las operaciones que los clientes pueden realizar sobre un objeto suelen declararse como métodos, que forman parte de la declaración de la clase. El paso de mensajes es una de las partes de la ecuación que define el comportamiento de un objeto; la definición de comportamiento también recoge que el estado de un objeto afecta a si mismo a su comportamiento. A diferencia de los elementos de datos pasivos en los sistemas tradicionales, los objetos tienen la posibilidad de actuar. La acción sucede cuando un objeto recibe un mensaje, que es, una solicitud que pide al objeto que se comporte de alguna forma. Cuando se ejecutan los

programas orientados a objetos, los objetos reciben, interpretan y responden a mensajes procedentes de los objetos. Por ejemplo, cuando un usuario solicita que un objeto llamado documento se imprima a sí mismo, el documento puede enviar un mensaje al objeto impresora solicitando un lugar en la cola de impresión; el objeto impresora puede devolver un mensaje al documento solicitando información de formato, y así sucesivamente. Los mensajes pueden contener información para clarificar una solicitud; por ejemplo, el mensaje solicitando que un objeto se imprima a sí mismo podría incluir el nombre de la impresora. Finalmente, el emisor del mensaje no necesita conocer la forma en que el objeto receptor está llevando a cabo la solicitud. En otras palabras, cuando el objeto documento recibe el mensaje imprimir, documento sabe exactamente lo que tiene qué hacer. El objeto que envía el mensaje ni sabe ni le importa cómo se realiza la impresión, solamente conoce que está sucediendo.

El conjunto de mensajes al que un objeto puede responder se llama protocolo del objeto. El protocolo para un icono puede constar de mensajes invocados por la pulsación del botón del ratón cuando el usuario localiza un puntero sobre un icono.

Los métodos pueden enviar también mensajes a otros objetos solicitando acción o información.

Al igual que las "cajas negras" de la ingeniería, la estructura interior de un objeto está oculta a usuarios y programadores. Los mensajes que recibe el objeto son los únicos contactos que conectan al objeto con el mundo exterior. Solamente un método del propio objeto puede disponer de los datos del interior del mismo para su manipulación. Estas características de los objetos confieren a la orientación a objetos su ventaja: La orientación a objetos fomenta la modularidad haciendo muy claras las fronteras entre objetos, explícita la comunicación entre los mismos y ocultos los detalles de la realización.

Cuando se ejecuta un programa orientado a objetos, ocurren tres sucesos. En primer lugar, se crean los objetos cuando se necesitan. Segundo, los mensajes se mueven de un objeto a otro (o desde el usuario a un objeto) a medida que el programa procesa internamente información o responde a la entrada del usuario.

Finalmente, se borran los objetos cuando ya no son necesarios y se recupera memoria.

3.1.3 Clases, subclases y objetos.

Muchos objetos diferentes pueden actuar de formas muy similares. Una clase es una abstracción que describe propiedades importantes para una aplicación y que ignora el resto. Una clase consta de métodos y datos que resumen las características comunes de un conjunto de objetos. La posibilidad de abstraer métodos y descripciones de datos comunes de un conjunto de objetos y almacenarlos en una clase es esencial para la potencia de la orientación a objetos . Definir clases significa situar código reutilizable en un depósito común en lugar de volver a expresarlo una vez y otra. En otras palabras, las clases contienen los anteproyectos para crear objetos. Finalmente, la definición de una clase ayuda a clarificar la definición de un objeto: un objeto es un modelo o instancia de una clase.

Los objetos se crean cuando se recibe un mensaje solicitando creación por la clase padre. El nuevo objeto toma sus métodos y datos de su clase padre. Los datos son de dos formas, variables de clase y variables modelo o de instancia. Las variables de clase tienen valores almacenados en una clase; las variables instancia tienen valores asociados únicamente con cada instancia u objeto creado a partir de una clase.

A título de ejemplo, considere cómo un programador podría designar una aplicación de procesamiento de textos en forma orientada a objetos. En primer lugar el programador identifica las entidades de interés. Los párrafos, por ejemplo, son objetos potenciales. Justificar es un método común para todos los párrafos. Tipodeletra (Fuente) es una variable de clase con el valor helvética. Finalmente, texto es una variable modelo con valores únicos para cada objeto. Es útil crear una clase llamada párrafo para guardar esta información común. La clase párrafo proporciona entonces un anteproyecto para la construcción de objetos. Aunque los datos específicos de cada objeto (ej.: las palabras del párrafo, el tipo de letra o

fuente, el interlineado o espaciado) pueden variar, todos los objetos de la clase párrafo comparten métodos y variables de clase comunes.

Una clase puede también resumir elementos comunes para un conjunto de subclases. En el caso del ejemplo del procesamiento de textos, el programador puede considerar después tabla para que sea otra clase además de párrafo. Pensando un poco más, el programador se da cuenta que párrafo y tabla comparten algunas propiedades con una clase más abstracta, texto. De forma similar, texto y gráfico pueden convertirse en subclases de una clase con carácter aún más general, documento ilustra esta jerarquía de las clases del procesamiento del documento. Utilizando subclases, los programadores orientados a objetos describen las aplicaciones como conjuntos de módulos generales o abstractos. Los métodos y datos comunes se elevan tan alto como sea posible de forma que sean accesibles a todas las subclases relacionadas.

A veces se denomina a las subclases como clases derivadas. En otras ocasiones los términos padre e hija se utilizan para indicar la relación entre una clase y una subclase. Las clases padre están localizadas por encima de las clases hijas en la jerarquía. Las clases más altas en la jerarquía se denominan las superclases.

Hasta ahora, esta descripción de un diseño orientado a objetos ha sido "ascendente". Es decir, la descripción pone el énfasis en la forma en que los componentes (objetos) son abstraídos para llegar a ser clases y superclases. De hecho, el término "descendente" ("top-down") describe con mayor precisión el método seguido por muchos programadores de la orientación a objetos. Normalmente comienzan su trabajo con una biblioteca de clases que contiene generalmente módulos útiles de programación. Utilizando clases predefinidas como punto inicial, los programadores escriben nuevas subclases que adaptan las clases de empleo general de la biblioteca a los requisitos funcionales particulares de la aplicación.

Una biblioteca de clases específicas para una aplicación se denomina un marco estructural "framework". Los marcos estructurales difieren de las bibliotecas de clases en su distinto grado: un marco estructural es una biblioteca de clases

ajustada especialmente para una determinada categoría de aplicaciones, por ejemplo, un marco estructural constructor de interfaces. Construir y adaptar aplicaciones a partir de marcos estructurales es más rápido y fácil que empezar con bibliotecas de clases genéricas. Asimismo, un marco estructural no será normalmente útil fuera del campo de la aplicación ya que contiene clases específicas para la aplicación.

3.1.4 Herencia.

La herencia es el mecanismo para compartir automáticamente métodos y datos entre clases, subclases y objetos. En términos generales se puede definir una clase que después se ira refinando sucesivamente para producir subclases. Todas las subclases poseen, o heredan, todas y cada una de las propiedades de su superclase y añaden además, sus propiedades exclusivas. No es necesario repetir las propiedades de las superclases en cada subclase. La herencia permite a los programadores crear nuevas clases programando solamente las diferencias con la clase padre. Cuando un programador declara que párrafo es una subclase de texto, por ejemplo, todos los métodos y variables modelo asociados con texto son heredados automáticamente por párrafo. Si la clase texto contiene métodos que son inadecuados para la subclase párrafo, entonces el programador puede obviar estos métodos escribiendo unos nuevos y almacenándolos como parte de la clase párrafo.

Debido a la herencia, los programas orientados a objetos constan de taxonomías, árboles o jerarquías de clases que, por medio de la sub clasificación, llegan a ser más específicas. Las clases proporcionan los anteproyectos para las subclases o para los objetos relacionados con una aplicación.

Herencia simple y múltiple son dos tipos de mecanismos de herencia utilizados normalmente en la programación orientada a objetos. Con la herencia simple, una subclase puede heredar datos y métodos de una clase simple así como añadir o sustraer comportamiento por sí misma. La herencia múltiple se refiere a la posibilidad de una subclase de adquirir los datos y métodos de más de

una clase. La herencia múltiple es útil al construir comportamiento compuesto a partir de más de una rama de una jerarquía de clases.

En resumen, los mecanismos básicos de la orientación a objetos conducen a una particular visión sobre el concepto de dar forma al mundo. Los elementos y su comportamiento se identifican como objetos. El comportamiento se realiza con métodos y datos almacenados en el objeto. Los mensajes obtienen el comportamiento de un objeto invocando un método del mismo.

Los objetos con métodos y variables modelo comunes se reúnen en una clase. Las clases se organizan en jerarquías y los mecanismos de herencia proporcionan automáticamente a cada subclase los métodos y datos de las clases padre. Las subclases se crean programando las diferencias entre las clases disponibles en una librería y los requisitos particulares de la aplicación.

3.2 Conceptos clave.

Los mecanismos básicos señalados anteriormente forman la base del paradigma de la orientación a objetos. Cuatro conceptos clave que resumen las ventajas del método orientado a objetos son la encapsulación, la abstracción, el polimorfismo y la persistencia.

3.2.1 Encapsulación.

La encapsulación (denominada también ocultamiento de información) consiste en separar los aspectos externos del objeto, a los cuales pueden acceder otros objetos, de los detalles internos de implementación del mismo, que quedan ocultos para los demás. La encapsulación evita que el programa llegue a ser tan interdependiente que un pequeño cambio tenga efectos secundarios masivos. La implementación de un objeto se puede modificar sin afectar a las aplicaciones que la utilizan. Quizá sea necesario modificar la implementación de un objeto para mejorar el rendimiento, corregir un error, consolidar el código o para hacer un

transporte a otra plataforma.

La encapsulación no es exclusiva de la programación orientada a objetos pero la capacidad de combinar la estructura de datos y el comportamiento en una única entidad hace que la encapsulación sea aquí mas limpia y potente que en los lenguajes convencionales que separan las estructuras de datos y el comportamiento.

3.2.2 Abstracción.

La abstracción consiste en centrarse en los aspectos esenciales inherentes de una entidad, e ignorar sus propiedades accidentales. En el desarrollo de sistemas esto significa centrarse en lo que es y lo que hace un objeto antes de decidir como debería ser implementado. El uso de la abstracción mantiene nuestra libertad de tomar decisiones durante el mayor tiempo posible evitado comprometernos de forma prematura con ciertos detalles. La mayoría de los lenguajes modernos proporcionan abstracción de datos pero la capacidad de utilizar herencia y polimorfismo proporciona una potencia adicional. El uso de la abstracción durante el análisis significa tratar solamente conceptos del dominio de la aplicación y no tomar decisiones de diseño o de implementación antes de haber comprendido el problema. Un uso adecuado de la abstracción permite utilizar el mismo modelo para el análisis, diseño de alto nivel, estructura del programa, estructura de una base de datos y documentación. Un estilo de diseño independiente del lenguaje pospone los detalles de programación hasta la fase final, relativamente mecánica del desarrollo.

3.2.3 Polimorfismo.

Los objetos actúan en respuesta a los mensajes que reciben. El mismo mensaje puede originar acciones completamente diferentes al ser recibido por diferentes objetos. Este fenómeno se conoce como polimorfismo. Con el polimorfismo un usuario puede enviar un mensaje genérico y dejar los detalles

exactos de la realización para el objeto receptor. El mensaje imprimir, por ejemplo, al ser enviado a una figura o diagrama invocará diferentes métodos de impresión que en el caso de enviar el mismo mensaje imprimir a un documento de texto.

El polimorfismo está fomentado por la maquinaria de la herencia. Es muy normal almacenar los protocolos para funciones de utilidad como "imprimir" en la posición más alta que sea posible dentro de la jerarquía de clases. Las variaciones necesarias en el comportamiento "imprimir" se almacenan en niveles más bajos de la jerarquía para sobrescribir los métodos más generales cuando sea necesario. De esta forma, los objetos están listos y pueden responder apropiadamente a mensajes de utilidad como "imprimir" mientras que el método que realiza la función de impresión puede existir en la clase inmediata del objeto o varios niveles por encima de la clase del objeto.

3.2.4 Persistencia.

La persistencia se refiere a la permanencia de un objeto, es decir, al tiempo durante el cual se asigna espacio y permanece accesible en la memoria de la computadora. En la mayoría de los lenguajes orientados a objetos, se crean modelos de clases a medida que el programa se ejecuta. Algunos de estos modelos se necesitan solamente por un breve período de tiempo. Cuando un objeto ya no es necesario, es destruido y recuperado el espacio de memoria que tenía asignado. La recuperación automática del espacio de memoria se denomina normalmente recolección de basura.

Después de haber ejecutado un programa orientado a objetos, los objetos ensamblados normalmente no se almacenan; es decir, los objetos dejan de ser persistentes. Una base de datos orientada a objetos mantiene una distinción entre objetos creados solamente para el tiempo de duración de la ejecución y aquellos pensados para almacenamiento permanente. Los objetos almacenados permanentemente se denominan persistentes.