1. Caso práctico

Unidad Didáctica II

Caso práctico

En esta unidad veremos una serie de conceptos básicos para la programación, tales como datos, tipos y estructuras de datos. Estos conceptos deben ser dominados por el programador a fin de incorporarse con ciertas garantías a un equipo de programación. Hay que especificar que los programas y aplicaciones informáticas trabajan con datos, que pueden ser almacenados para posteriormente ser utilizados en la obtención de información útil, empleada para adoptar las decisiones más adecuadas en cada situación.

CASO. Nuestros amigos han comprobado que una aplicación informática es un trabajo de equipo, que debe estar perfectamente coordinado y en el que se habla el mismo idioma, es decir, utilizar siempre la misma terminología y conceptos para referirse a las diferentes situaciones que se van presentando. Especialmente ***Victor*** *que no era muy partidario de estudiar y piensa que todo siempre es más fácil de lo que parece.*
Recuerda una ocasión, cuando tuvieron que trabajar con una empleada de la empresa ***Servicios Locales S.L.****,* ***Cristina****, cuya tarea era la de revisar el trabajo y comprobar que la aplicación se adaptaba a las especificaciones del cliente. Ahí se percató de que no se enteraba de las conversaciones al no dominar la terminología informática, por lo que se le hacía necesario conocer algunos conceptos básicos de programación de ordenadores.*

Datos: Tipos y Características

2. Concepto de Dato

Unidad Didáctica II

Concepto de Dato

CASO. Una situación muy común en los equipos de programación es la definición de los datos y variables que se van a utilizar. Es posible que sea la parte más delicada de la programación ya que condicionará el desarrollo de todo el trabajo. En ocasiones optar por un tipo de dato puede suponer mejorar el rendimiento del programa y un ahorro de espacio en los sistemas de almacenamiento.

*Nuestros amigos de **Soluciones Informáticas Andalucía, S.C.A.** (en adelante **SI Andalucía**) saben que para definir los datos a utilizar, es preciso tener una visión global de la aplicación y conocer perfectamente a lo que se van a enfrentar. En concreto, ahora están trabajando en una aplicación para **Servicios Locales S.L.** en la que tienen que procesar los datos de los empleados de la empresa y de sus clientes, (nombres, apellidos, teléfono, dirección, edad, sueldo de cada mes, IRPF, etc.) y tienen que decidir qué tipos de datos serán más adecuados para cada caso.*

*A lo largo de la unidad **Víctor**, y tú con él, aprenderéis a qué nos referimos al hablar de **datos**, qué datos concretos debemos considerar para la aplicación que deben desarrollar, y los criterios que deben usarse para elegir el tipo más adecuado para cada dato. Debéis conocer todas las opciones sobre los datos y cuales son las condiciones óptimas para utilizar un dato u otro o una estructura de datos u otra.*

Como ya **vimos** en la unidad 1, podemos definir **dato** como un conjunto de símbolos que representan valores, hechos, objetos o ideas de forma adecuada para ser **tratados**. Incluso, en el ámbito de la informática, podemos definir dato de forma similar, como cualquier "objeto" manipulable por la computadora.

Un dato puede ser un carácter leído desde teclado, un número, la información almacenada en un CD, una foto almacenada en un fichero, una canción, el nombre de un alumno almacenado en la memoria del ordenador, etc.

Los ordenadores son máquinas creadas para procesar automáticamente la información, y para ello esa información no se almacena ni representa al azar, sino que ha de **organizarse y estructurarse de forma adecuada para que pueda almacenarse, procesarse y recuperarse de la forma más eficiente posible**. Éste será el principal problema del que nos ocuparemos en esta unidad.

En esta unidad analizaremos:

- **Los tipos básicos de datos que incluyen la mayoría de los lenguajes de programación**, (aunque con diferencias significativas entre unos y otros) .
  Por ejemplo, la edad de un trabajador la podremos representar con un tipo básico, concretamente un tipo **entero**, ya que la edad es un valor numérico positivo y sin decimales (que es justo lo que define el tipo entero, como se verá más adelante).

- **Las estructuras de datos que pueden crearse a partir de ellos**.
  Por ejemplo, todos los datos relativos a un trabajador (nombre, edad, sueldo, etc.) formarán

una estructura de datos llamada **registro**, y el conjunto de todos los registros de los trabajadores se almacenará en otra estructura, que llamamos **fichero o archivo,** y estos a su vez se pueden agrupar en **Bases de Datos**. Como vemos, las estructuras se forman a partir de los tipos elementales, pero también a partir de otras estructuras.

- **La utilidad de estas estructuras**, aunque no se abordará su estudio en profundidad hasta unidades posteriores, una vez comenzado el estudio del lenguaje Java.
  Por ejemplo, el fichero será útil para almacenar los datos de los trabajadores de forma permanente, de esta forma conseguiremos conservarlos una vez que al apagar el ordenador se borre la memoria RAM.

Datos: Tipos y Características

3. Tipos de datos

Unidad Didáctica II

Tipos de datos

*CASO. **Victor** ha comprendido, con la ayuda de **Carmen**, que es interesante conocer todas las posibilidades y opciones para adoptar la mejor según las circunstancias del momento y las necesidades del cliente. Recuerda un caso en el que para almacenar la fecha de nacimiento de un empleado, él decidió almacenar el dato como un texto, con el fin de poder guardar los guiones de separación del dia, mes y año.*
*En aquel momento **Carmen** le dijo que hacerlo así le traería más problemas ya que es necesario comprobar que la fecha es correcta (por ejemplo que no se almacene un 30 de febrero o un 31 de noviembre). **Víctor** decide entonces hacer un pequeño programa que soluciona la situación, **Carmen** le comenta que efectivamente no es complicado pero que debe tener en cuenta los años bisiestos. Finalmente le hace ver que si define el dato como de tipo "fecha", se ahorraría todo el trabajo, ya que con este tipo el compilador se encarga de comprobar gran parte de las situaciones anómalas y el programa sería más eficiente.*

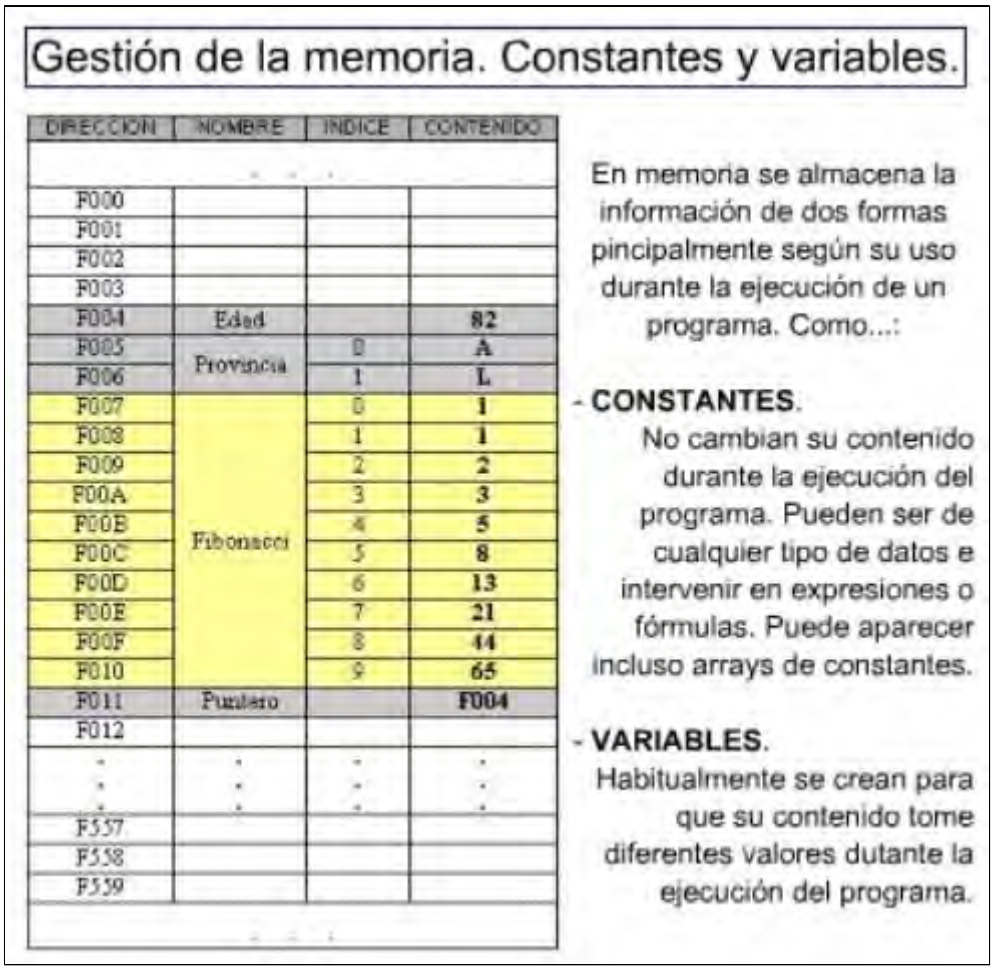
Sitúate en el caso de la aplicación a desarrollar para la empresa Proyectos Locales, S.L., ¿piensas que todos los datos tienen siempre el mismo valor, o por el contrario serán modificados alguna vez? Seguramente tendremos datos que no cambiarán nunca, como el **CIF** de la empresa (código de identificación fiscal), que puede considerarse un valor bastante "constante", ya que se introduce en la aplicación una vez, y nunca va a ser necesario modificarlo. Las empresas no cambian de CIF. Otros datos sin embargo cambiarán de valor con muchísima frecuencia, como la hora expresada con precisión de minutos que debe aparecer en una esquina de la ventana de nuestra aplicación. Variará su valor cada minuto, por lo que puede considerarse un dato bastante "variable".

Lo mismo ocurre en todos los programas. Usaremos datos asociados normalmente a **constantes** y a **variables**.

Tanto constantes como variables son zonas de la memoria del ordenador a las que se le asigna un nombre (identificador), a modo de etiqueta, y a las que se les asocia un tipo de dato, de forma que sólo se podrán almacenar en esa zona de memoria (o posición de memoria) datos de ese tipo.

La diferencia:

- **Las constantes reciben valor una vez**, normalmente al principio del programa, **y ya no puede volver a escribirse nada en la posición de memoria que ocupan** (una vez que se les asigna el valor, no se puede modificar, permanece constante).
- **Las variables pueden cambiar de valor tantas veces como se desee a lo largo del programa** (podemos volver a escribir otro valor en la zona de memoria que ocupan tantas veces como deseemos).



En definitiva, **para usar los datos, tenemos que disponer de ellos en la memoria del ordenador**. Es como si la memoria fuese un enorme casillero, en el que cada casilla puede almacenar un dato, y tiene una dirección asignada (realmente un **número binario**) y puedo referirme a una casilla en concreto y al valor que contiene a través de su dirección.

Así se hace de hecho cuando se programa en código máquina o ensamblador, pero eso es sumamente dificultoso, y requiere conocimiento de los detalles de la arquitectura del ordenador, de su funcionamiento interno, que no queremos manejar nosotros. Por eso, **en los lenguajes de alto nivel, se asocia un nombre de variable o constante con una de estas direcciones de memoria. Cada vez que en el programa hagamos mención al nombre de la variable o constante, el compilador sustituirá ese nombre (esa "etiqueta") por la dirección de memoria (en binario) que tenga asociada**. Para nosotros es mucho más fácil de entender, por ejemplo, que a la variable "***edad***" se le suma 1 que entender que al dato almacenado en la posición de memoria 1000 1100 1010 1010 0100 1110 se le suma 1. Es muchísimo más descriptivo y más claro usar **el nombre de la variable, al que llamamos identificador**, que la dirección de memoria asociada.

Pero también hemos mencionado que **la variable (o constante), además del nombre lleva asociado un tipo de dato.** ¿Qué es eso? **Un tipo de dato no es más que una especificación de los valores que son válidos para esa variable y de las operaciones que se pueden realizar con ellos.**

Datos: Tipos y Características

3.1. Preguntas que deben definir un tipo de datos

Unidad Didáctica II

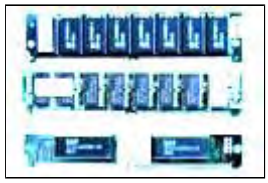
Preguntas que deben definir un tipo de datos

Un tipo de datos contesta a dos preguntas:

- **¿Qué valores o datos son válidos?**
- **¿Qué operaciones puedo hacer con esos valores o datos?**

Por ejemplo, el tipo entero permitirá unos valores y unas operaciones distintas a las del tipo real.

*Si el sueldo de un empleado de **Servicios Locales, S.L**. lo almacenáramos en una variable de tipo entero, llamada sueldoMensual, no podría contener decimales, por lo que debería ser una cantidad exacta de Euros, que no admitiera céntimos, lo cual, al precio que está la vida, podría tener sentido. 99 céntimos más o menos en el sueldo de un mes son despreciables para el trabajador, y un ahorro para la empresa si tiene muchos empleados. De hecho **María** y **José** al hacer el análisis de la aplicación pensaron que esto sería así en la realidad, y definieron sueldoMensual como una variable de tipo entero. Pero como en **Servicios Locales S.L**. son muy legales, y no les gusta quedarse con lo que no es suyo, exigen a **Soluciones Informáticas Andalucía, S.C.A.** que corrijan ese fallo, y que la aplicación permita introducir los céntimos en la nómina del trabajador. No hay problema. **José** le dice a **Carmen** que cambie el tipo de la variable sueldoMensual, que deberá ser ahora real.*



*Por otro lado, **José** se tienen en cuenta que al calcular el sueldo mensual de cada trabajador, se divide el sueldo anual entre 14 (doce meses más dos pagas extras). Necesita que esa operación se realice con precisión de al menos dos decimales para indicar los céntimos del sueldo mensual. Si ambos valores son de tipo entero, la operación a realizar es la división entera, (sin obtener decimales), por lo que el valor obtenido no se ajustará a la realidad. Es preciso definir el sueldo anual como de tipo real para poder realizar la división real, que sí es una operación disponible para los números reales, y que sí obtiene los decimales, para así poder expresar con precisión de céntimos el sueldo mensual. ¡**Carmen**, modifica también la variable sueldoAnual, para que también sea real!*

DEMO: Consulta esta simulación para entender un poco mejor el almacenamiento de la información en la memoria y las principales diferencias entre los diferentes tipos de datos.

A continuación veremos algunos de los **tipos** existentes en la mayoría de los lenguajes de programación. NO todos los lenguajes disponen de los mismos tipos de datos. Hay lenguajes más ricos que otros en cuanto a tipos de datos disponibles. Aquí describiremos brevemente los más usuales, y será con la introducción al lenguaje Java cuando nos familiarizaremos con los que incluye ese lenguaje.

Datos: Tipos y Características

4. Datos Simples o básicos o primitivos

Unidad Didáctica II

Datos Simples o básicos o primitivos

CASO. Al hacer programas, la materia prima con la que se trabaja son los datos, los cuales se utilizan en expresiones con los operadores adecuados para obtener los resultados deseados. En la empresa de nuestros amigos tienen muy claro que conocer los datos con los que van a trabajar, supone una gran ayuda en lo que a eficiencia se refiere.

***Carmen** sabe que la experiencia a la hora de programar suele ser determinante, pero recuerda que sus profesores le comentaban que la elección errónea de los tipos de datos supone que el programa no funcione en todos los casos. Un profesor de programación le explicó en una ocasión que los tipos de datos fueron creados por los programadores según las necesidades que iban encontrando al programar sus aplicaciones.*

Realmente deberíamos hablar más de tipos simples de datos que de datos simples. Usamos la segunda nomenclatura por simplicidad.

¿Qué son los datos simples o primitivos? ¿Acaso son datos poco perfeccionados, o que no puedan usarse para cálculos complicados?

No exactamente. **Son los datos que suele proporcionar el lenguaje de programación, y que por eso de ellos conocemos:**

- **Cómo se representan internamente (en memoria).**
- **El tamaño exacto que ocupan.**
- **Los operadores que define el lenguaje para ellos.**
- **Que están disponibles sin necesidad de definir nada por parte del usuario.**

Son un conjunto mínimo de tipos de datos, de forma que a partir de ellos se construirán los demás tipos de datos, que por ello recibirán el nombre de datos estructurados.

4.1. Cuadro de tipos de datos primitivos en java

Unidad Didáctica II

Cuadro de Tipos de Datos Primitivos en Java

Tipo	Descripción
boolean	Permite representar valores lógicos; Verdadero (**V**) o Falso (**F**).
char	Permite representar un símbolo o carácter UNICODE de 16 bits.
byte	Entero de **8 bits** con signo (representado en complemento a dos). Su rango de valores va desde **-128 (-2^7)** a **+127 ($+2^7-1$)**.
short	Entero de **16 bits** con signo (complemento a dos). Rango de valores entre **-32.768 (-2^{15})** y **+32.767 ($+2^{15}-1$)**.
int	Entero de **32 bits** con signo (complemento a dos). Rango de valores entre **-2.147.483.648 (-2^{31})** y **+2.147.483.647 ($+2^{31}-1$)**.
long	Entero de **64 bits** con signo (complemento a dos). Rango de valores entre **-2^{63}**y **$+2^{63}-1$**.
float	Número real (en coma flotante) de 32 bits, utilizando la representación IEEE 745-1985.
double	Número real (en coma flotante) de 64 bits, utilizando la representación IEEE 745-1985.

Datos: Tipos y Características

4.2. Ejemplos de tipos de datos

Unidad Didáctica II

Ejemplos de tipos de datos

*Mientras **Carmen** y **Víctor** trabajan en la aplicación, éste tiene algunas dudas sobre el tipo adecuado para cada variable que forma parte del registro de cada empleado, y le dice a **Carmen** que un ejemplo, de alguna variable de cada tipo básico le sería muy útil para decidirse por un tipo o por otro. **Carmen** piensa que a fin de cuentas, no le llevará mucho tiempo buscar un ejemplo, y le hace en un momento la siguiente lista:*

Dato	Descripción	Tipo	Breve justificación
Casado	Queremos saber si el trabajador está casado o no, a fin de elaboración de horarios que faciliten la conciliación de la vida laboral y familiar	**boolean**	Sólo son posibles dos respuestas: Sí o No (Verdadero o Falso)
Sexo	Queremos almacenar el sexo del empleado, y distinguir entre Hombre, Mujer y Desconocido, ya que existe la posibilidad de que el trabajador sea extranjero, con un nombre que no nos deje claro su sexo, y que haya olvidado indicarlo al rellenar la ficha de sus datos personales.	**char**	Con una sóla letra podemos distinguir entre las 3 situaciones posibles, asignando 'H' a Hombre, 'M' a mujer y 'D' a Desconocido
Dia de la semana	Se expresará como número del 1 al 7, pero tenemos que tener presente que hay problemas de capacidad de memoria para el ordenador en que se va a ejecutar la aplicación, por lo que debemos ahorrar todo el espacio de almacenamiento que podamos	**byte**	Permite representar números enteros entre -128 y 127, por lo que es más que suficiente para los valores de 1 a 7 que queremos guardar. Además sólo ocupa 8 bits. Es el tipo entero más pequeño que podemos usar.
Dia del año	Se expresará como un número entre 1 y 366. Al igual que antes, queremos aprovechar al máximo el espacio de almacenamiento.	**short**	Permite representar números enteros entre -32.768y 32.767, por lo que es más que suficiente para los valores de 1 a 366 que queremos guardar.Sin embargo el tipo byte no proporciona un rango suficiente. Además sólo ocupa 16 bits. Es el tipo entero más pequeño que podemos usar para este conjunto de valores.
Total Factura	Será una cantidad redondeada a euros, ya que nuestros clientes nos pagan grandes sumas, en las que incluir céntimos parece ridículo.	**int**	Permite representar números enteros, sin decimales, entre -2.147.483.648 y 2.147.483.647. Por mucho que nuestros clientes nos paguen grandes sumas, no tendremos la suerte de que nos paguen más de dos mil millones de euros en una sola factura.
Milisegundos	Número de milisegundos que han trascurrido desde las 00:00:00 horas del día 1	**long**	Es una cantidad considerable, que no podemos almacenar en un int, pero que sigue siendo un número

	de Enero de 1970 hasta nuestros días.		entero en el rango de long. De hecho esa es la cantidad que usa Java en la clase Date para expresar una fecha.
Sueldo Mensual	Queremos expresar el sueldo de cada empleado en euros, con precisión de céntimos. Suponemos que seguimos con limitaciones de memoria.	**float**	Es el tipo de número real (con decimales) de menor precisión y que menos memoria ocupa. No obstante, su rango de valores permite representar el sueldo de cualquier trabajador de la empresa, por elevado que éste sea. Además tampoco necesitamos más precisión, ya que para los céntimos, con dos decimales nos basta.
π	Número Pi. Necesitamos almacenar el valor del número pi para una aplicación que realiza cálculos científicos de gran precisión. Necesitamos poder representar el mayor número posible de decimales del número Pi, para no perder precisión en los cálculos.	**double**	Es el tipo real que mayor precisión nos proporciona. El número Pi es un número real irracional, es decir, con infinitos decimales. Con double podremos representar de forma exacta el mayor número posible de ellos.

Datos: Tipos y Características

4.3. Numéricos

Unidad Didáctica II

Numéricos

*CASO. Al hacer un programa **Víctor** vió que aparecían variables de diferentes tipos para almacenar valores numéricos y comentó que si en definitiva todas esas variables almacenaban números, bastaría con un tipo de dato numérico que admitiera decimales y no sería necesario distinguir entre enteros y reales. **José** le explicó que no todos los números se utilizan del mismo modo y que hay operaciones que requieren mayor precisión, por ese motivo se decidió, en los albores de la programación, diferenciar varios tipos de datos numéricos, cada uno de ellos con unas características diferentes. No es lo mismo guardar la edad de una persona en años, que guardarla en segundos. También es diferente calcular el peso de un avión y el de un electrón.*

Seguro que una de las primeras cosas que aprendiste a hacer en el colegio, fue resolver problemas usando operaciones matemáticas: las famosas "cuatro reglas" de sumar, restar, multiplicar y dividir. Resultaría realmente difícil resolver la mayoría de los problemas sin la inestimable ayuda de los números y las matemáticas, ¿no? Por eso es casi impensable que los ordenadores, que pretendemos usar para resolver todo tipo de problemas, y además de forma automática no usaran números y operaciones matemáticas. Por eso es imprescindible que cualquier lenguaje defina como tipos básicos algunos tipos de datos numéricos.

Los tipos **numéricos** son tipos básicos que nos permiten representar valores de tipo numérico. La mayoría de los lenguajes sólo permiten representar y usar dos grupos de datos numéricos, para números enteros y para números reales. Cualquier otro tipo numérico como por ejemplo los números Racionales o los números Complejos, deberán ser definidos por el usuario como una estructura a partir de los tipos simples entero y real. Así, por **ejemplo**, un número racional será un par de números enteros. Un número complejo se definirá como un par de números reales. Esa definición debe hacerla el programador de la aplicación, y también deberá programar las operaciones permitidas para cada uno de esos tipos. No obstante, en la mayoría de las aplicaciones nos basta y nos sobra con los tipos numéricos que ya nos proporciona el lenguaje.

En el ejemplo de la aplicación de nuestros protagonistas, necesitaremos almacenar en variables de tipo numérico la edad del trabajador, el sueldo anual o el sueldo mensual, por ejemplo.

Datos: Tipos y Características

4.4. Enteros. Tipos de enteros.

Unidad Didáctica II

Enteros. Tipos de enteros

*CASO. **Carmen** recuerda una ocasión en la que con un programa de sumas de números positivos a veces se encontraba con resultados negativos y por más que revisaba el código no encontraba errores, cuando estudió detenidamente los tipos de datos numéricos entendió el error que estaba cometiendo.*
El problema era que utilizaba números definidos como 'short' y algunas veces el resultado excedía el valor límite máximo de 32.767 para este tipo y el resultado que devolvía la memoria era incoherente al no poder almacenarlo. Lo que ocurre es que los números enteros usan una aritmética cíclica.

¿Qué conjunto de números fue el primero que estudiaste en la escuela, cuando aprendías a contar? Si recuerdas bien, fueron los número **Naturales**, del cero en adelante. Pero rápidamente aprendiste que había operaciones como la resta que no siempre podían tener solución dentro de los naturales, así que se era necesario buscar un conjunto más grande, el de los números Enteros, que contenía a los Naturales, pero también a los números negativos. Y con ese conjunto ya eras capaz de resolver gran cantidad de problemas, ¿no?

Los datos de tipo entero permiten representar números enteros (sin decimales) tanto positivos como negativos. La mayoría de los lenguajes definen varios tipos de enteros, que se diferencian fundamentalmente en el número de **bytes** que se usan para su representación. Mientras más bytes usemos para representar un número entero, mayor será el rango de números representable.

Por ejemplo, en Java, existen cuatro tipos de números enteros:

Nombre del tipo entero	Tamaño usado para su representación (bits)	Total de números distintos representables	Menor número representable para el tipo	Mayor número representable para el tipo
byte	8 =1 byte	$2^8 = 256$	-128	+127
short	16 =2 bytes	$2^{16} = 65.536$	-32.768	+32.767
int	32 = 4 bytes	$2^{32} = 4.294.967.296$	-2.147.483.648	+2.147.483.647
long	64 = 8 bytes	$2^{64} = 1,844674407 E+19$	-9.223.372.036.854.775.808	9.223.372.036.854.775.807

Naturalmente otros lenguajes definirán otros datos de tipo entero, pero es usual que al menos todos los lenguajes dispongan de un tipo "entero" y de un tipo "entero largo". Las diferencias entre unos tipos enteros y otros, serán siempre el rango de valores representables.

Para saber más:
Existen diferentes maneras de representar internamente los distintos tipos de datos, si quieres profundizar un poco más en este tema, visita el siguiente enlace, en la página 7 puedes encontrar la notación en complemento a 2 para representar los números enteros:
Representación interna de tipos
http://www.conectados.org/doc/Apuntes/1_Elementos_de_Programacion/temai.pdf [Versión en caché]

(Si tienes problemas para acceder a algún enlace, pulsa en "Versión en caché" para visualizar una copia de esa página

web)
Para descargar el programa Acrobat Reader pulsa aquí.

¿Cuando definiremos una variable como de un tipo u otro? Dependerá del tipo de datos que manejemos en nuestro problema. Si conocemos que el valor máximo o mínimo de nuestros datos nunca va a sobrepasar el máximo o el mínimo de un tipo, definiremos la variable de ese tipo. Mientras menor sea el número de bits que usa la representación del tipo, menos memoria ocupa, y por tanto mejor estaremos aprovechando los recursos del ordenador. Hay que pensar que en principio la memoria es siempre un bien escaso en un ordenador.

Como **ejemplo** de cuándo elegir un tipo entero u otro, puedes volver a mirar la tabla de ejemplos del apartado "Datos Simples o básicos o primitivos" de esta misma unidad, que **Carmen** le hacía a **Víctor** para aclararle las ideas.

No obstante, en **la actualidad, la diferencia entre definir todas las variables de un programa por ejemplo de tipo short o definirlas de tipo int puede ser una cantidad de memoria insignificante o al menos no relevante para el rendimiento de la aplicación. Es por ello usual definir las variables y constantes enteras directamente como int, aunque sepamos que su valor nunca va a ser mayor de 10, por ejemplo**. Sin embargo **sólo definiremos las variables o constantes como long cuando sepamos con seguridad que vamos a manejar datos o valores realmente grandes, ya que 8 bytes (64 bits) empiezan a ser un tamaño no despreciable**. En la mayoría de las aplicaciones de gestión, rara vez tendremos necesidad de usar los tipos de enteros largos (Por ejemplo, long) que sí tendrán su utilidad en aplicaciones que realicen cálculos científicos.

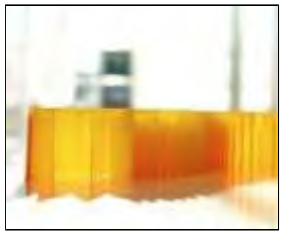
Datos: Tipos y Características

4.5. Consideraciones sobre los tipos enteros

Unidad Didáctica II

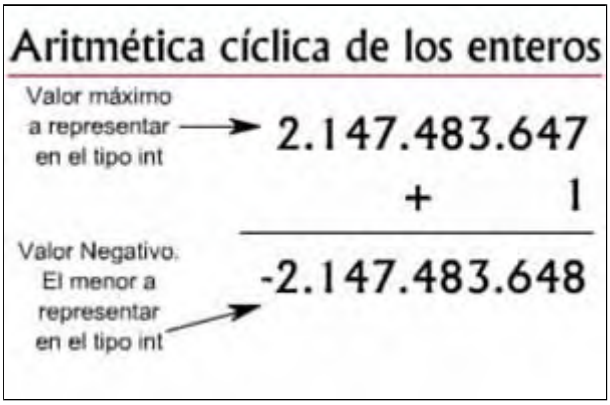
Consideraciones sobre los tipos enteros

Por ahora debes tener claro lo que es un tipo de dato, lo que es un tipo numérico, los distintos tipos de datos enteros que puede haber, sus diferencias y cuando elegir uno u otro. Pero no todo son diferencias, ¿verdad? Sea cual sea el tipo entero con el que trabajes, deberás tener presente que todos tienen en común, entre otras, las siguientes características:



- Los **operadores** que suelen tener asociados los tipos enteros son los operadores matemáticos usuales: **suma, resta, división entera (sin sacar decimales), multiplicación y operación resto-módulo (el resto de la división entera)**.
- **Sea cual sea el tipo elegido, siempre habrá números enteros que no se podrán representar**, ya que el conjunto de los números enteros es infinito. A pesar de ello, si lo necesitamos, podríamos hacer un programa que realizara operaciones con números más grandes, pero seríamos nosotros los que deberíamos programar esas operaciones.
- **Tienen una "aritmética circular",** de forma que **si al número más grande que se puede representar para un tipo entero le sumo 1, no se produce ningún error, sino que se obtiene el número más pequeño (más negativo) que se puede representar para ese tipo**. **Igualmente, si al número más pequeño que se puede representar se le resta 1, se obtiene el número más grande que se puede representar para ese tipo.**

Así, por **ejemplo**, para una variable de tipo int en Java, si tiene el valor **2.147.483.647** (valor máximo representable para int) y le sumamos **1** nos da cómo resultado **-2.147.483.648**.



De igual modo si a **-2.147.483.648** (el menor valor representable) le restamos **3** obtendríamos **2.147.483.645** (el mayor valor representable menos 2).

Autoevaluación

Señala la afirmación correcta:

a) El operador división entera consiste en división con decimales hasta que cociente es cero.

b) La operación resto-módulo es el resto de la división entera.

c) Podremos representar todos los números reales. Para ello sólo tendremos que incrementar el número de bits que usaremos para su representación.

d) Todas las anteriores son correctas

Comprobar

Datos: Tipos y Características

4.6. Reales. Tipos de reales

Unidad Didáctica II

Reales. Tipos de reales

*CASO. Cuando el equipo de programación se enfrenta al programa de cálculo de nóminas de la empresa **Servicios Locales**, debe diferenciar entre valores numéricos enteros y reales. La mayor parte de los resultados van a tener cifras decimales que se pierden si se guardan en variables de tipo entero. **Víctor** pide entonces que todos los resultados sean almacenados en variables reales, **Carmen** le explica que sencillamente las variables de tipo entero requieren menos memoria que las de tipo real.*

Al igual que en el colegio aprendiste a solucionar problemas que hicieron necesaria la introducción de los números negativos, pronto se vió la necesidad de incluir más números, ya que había divisiones de números enteros que no tenían solución dentro de los números **enteros**. Aparecieron los **racionales**, los números con decimales, en definitiva. Sin pretender dar una clase de matemáticas, al conjunto de todos los números con decimales, ya sea un número finito o infinito de ellos, se le llamaba **reales**, y prácticamente nos permitían solucionar todos nuestros problemas de la vida corriente. ¿Concibes un lenguaje que no te permitiera representar números con decimales? No podríamos facturar 1,5 kg de patatas, un chicle no podría costar menos de un euro, o el IPC no podría ser del 3,9 por ciento, o el peso de un electrón tendría que ser de al menos un gramo para poder representarlo, por ejemplo.

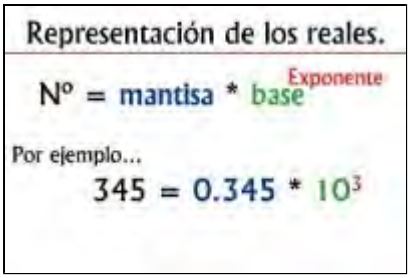
Por eso también son necesarios los **datos de tipo real** en los lenguajes de programación.

Los datos de tipo real se usan para representar números reales (con decimales). Al igual que ocurre con los enteros, la mayoría de los lenguajes definen más de un tipo de datos real, que se diferencian igualmente entre sí por el número de bits que se usan para representarlos y almacenarlos.

Mientras más bits usemos, podremos representar números:

- **Más grandes en valor absoluto** (tanto positivos como negativos)
- **Con mayor precisión** (con más decimales exactos)

Sabemos que entre cada dos números reales cualesquiera siempre tendremos infinitos números reales. Evidentemente, al disponer de un número finito de bits, **no podemos representar todos los infinitos números reales, por lo que la mayoría de ellos los representaremos de forma aproximada.**



Los números reales se representan en coma flotante, que viene a ser algo parecido a notación exponencial o científica, pero con algunas diferencias para poder representar el máximo de números posible.

Un número se expresa como: Nº = mantisa * Base^{exponente}.

En concreto, sólo se almacena la **mantisa**, que son las cifras decimales significativos, y el exponente al que va elevada la base. El exponente se ajusta de forma que la parte entera del número siempre sea cero, y por tanto la parte entera no se almacena.

La base de numeración, como también será siempre la misma, tampoco se almacena. Los algoritmos que operan con datos reales, tienen en cuenta cuales son esos valores, y las operaciones se realizan

correctamente.

Para saber más:
Al igual que con los números enteros, los números reales tienen diferentes representaciones internas, entre ellas está la representación interna en coma flotante, en la notación IEEE 754, de la que podrás obtener más información en el siguiente enlace:
Representación interna de los números
http://www.unalmed.edu.co/~daristiz/guias/fisica_computacional/semana_4/aritmetica.pdf
[Versión en caché]

(Si tienes problemas para acceder a algún enlace, pulsa en "Versión en caché" para visualizar una copia de esa página web)
Para descargar el programa Acrobat Reader pulsa aquí.

Datos: Tipos y Características

4.7. Características de la representación de números reales

Unidad Didáctica II

Características de la representación de números reales

El hecho de que no se pueda representar cualquier número de forma exacta, plantea algunos inconvenientes. Por ejemplo, **no siempre se cumple la propiedad asociativa ni la distributiva cuando operamos con las representaciones de números reales (aunque en teoría sí deben cumplirse).** El orden en que se realizan las operaciones influye en el resultado, ya que en cada operación se producen errores por falta de precisión en la representación (errores de aproximación) que se van acumulando.

DEMO: Pulsa aquí para ver un ejemplo en el que no se cumple la propiedad asociativa.

Ejemplo: Supongamos por simplificar un sistema de representación con base 10 y 4 dígitos decimales para la mantisa.
X = 5000 = .5 *10^4
Y = - 4999 = - .4999 * 10^4
Z = 1.5 = .15*10^1 = .00015 * 10^4 ≈.0001 * 10^4
Z+X)+Y = .5001 * 10^4 - .4999 * 10^4 = .0002 * 10^4 = 2,
que debería ser igual a
Z+(X+Y) = .15 * 10^1 + (.5 *10^4 - .4999 * 10^4) = .15 * 10^1 + .0001 * 10^4 = .15 * 10^1 + .1 * 10^1 = .25 * 10^1 = 2.5 , que no es 2,
con lo que parece que no se cumple la propiedad asociativa.

- **Tampoco siempre se cumple propiedad distributiva**, o al sumar y restar el mismo valor, si se trata de un valor grande que obliga a desechar cifras del resultado. Con los mismos datos del ejemplo anterior se observa que:
  Z* (X+Y) = 1.5 *1 = 1.5 y sin embargo **(Z * X)+ (Z * Y)** = 7500 - 7498 = 2 , ya que al multiplicar Z por Y **se obtienen cinco cifras significativas, pero solo podemos representar cuatro**.

- **Tambien ocurren algunos problemas de incoherencia en expresiones.**
  (X+Z)-X = (.5* 10^4 + .0001 * 10^4) - .5 * 10^4 = .5001 * 10^4 -.5 * 10^4 = .0001 * 10^4 = 1, **que no es Z**.

- **Cuando al operar con dos números reales obtenemos un resultado mayor que el mayor número representable, o menor que el menor nº representable (más negativo) se produce un error de overflow** (desbordamiento por arriba o por grande) que hace que falle el programa.

- **Si al operar obtenemos un resultado más próximo a cero que el número más cercano a cero que se puede representar (tanto en positivo como en negativo), se produce un error de underflow** (desbordamiento por abajo o por pequeño) que también hace que falle el programa.

- **La mayoría de los lenguajes definen también como tipos básicos (o simples o primitivos), más de un tipo de números reales, que se diferencian en el número de bits usados para la representación, y por tanto en la precisión de los números representados.**

Por ejemplo, en Java se definen como de tipo real los siguientes:

Nombre del tipo real	Tamaño usado para su representación (bits)	Total de números distintos representables	Menor número representable para ese tipo (en valor absoluto)	Mayor número representable para ese tipo (en valor absoluto)

float	32 = 4 bytes	2^{32} = 4.294.967.296	1.401298464324817 E-45	3.4028234663852886 E+38
double	64 = 8 bytes	2^{64} = 1,844674407 E+19	4.9 E-324	1.7976931348623157 E+308

En el caso de los números reales, cuando no tengamos limitaciones serias de la memoria disponible para ejecutar nuestra aplicación, **se recomiendausar siempre el tipo de mayor precisión**, ya que así, al operar, **los errores cometidos por las sucesivas aproximaciones serán menores**. De hecho, en Java los literales reales se consideran double por defecto, y se recomienda que todas las variables de tipo real se definan como double. La diferencia entre usar 8 bytes para cada variable double en vez de los 4 bytes de float en un programa que use 10.000 variables de tipo real (lo cual es muchísimo) supone usar 80.000 bytes en vez de 40.000, lo que supone una diferencia de menos de 40 Kbytes de memoria. Cuando hoy en día los ordenadores suelen tener ya incluso más de un Gigabyte de **memoria RAM**, 40 Kbytes se pueden considerar bien empleados a cambio de mejorar la precisión de los cálculos.

Los operadores disponibles para los tipos reales son las operaciones matemáticas usuales: **suma, resta, multiplicación, división real (con decimales)**

Datos: Tipos y Características

4.8. Ejemplo de cómo elegir un tipo real u otro

Unidad Didáctica II

Ejemplo de cómo elegir un tipo real u otro

Nuestro amigo **Víctor** andaba rompiéndose la cabeza **calculando el rango** posible para los datos de un par de variables reales que recibían su valor tras complejos cálculos matemáticos, **y también la precisión** necesaria para representar los valores de la forma más exacta posible. Quería saber qué tipo de dato sería necesario usar para aprovechar al máximo la capacidad de memoria. **María**, que lo veía muy concentrado le preguntó:

- **María**: Sabes que las variables deben almacenar decimales, y sabes que entonces sólo pueden ser float o double ¿Cuántas variables te está planteando el dilema?
- **Víctor:** Dos.
- **María**: Pues eso supone que si usas **float**, estarás usando 8 bytes, y si usas **double** estarás usando 16 bytes. La diferencia de 8 bytes es el tamaño de memoria que se necesita para guardar 4 letras en Java. No parece que elegir double sobrecargue la memoria. Usa double sin mirar nada más, y estarás garantizando la máxima precisión en los cálculos.
- **Víctor:** Pero es que una de ellas se almacena para cada uno de los 8 millones de registros de una base de datos, que además está en un disco duro sin demasiado espacio libre.
- **María:** Eso cambia las cosas. ¿Necesitas mucha precisión para esa variable?
- **Víctor:** No, sólo 4 cifras decimales.
- **María:** En ese caso, mejor define como double la otra variable, y ésta como float. Estarás ahorrando unos 30 Megabytes de espacio de almacenamiento en disco. Pero de todas formas, recomiéndale a la empresa que deberían plantearse comprar un disco duro más grande.

Datos: Tipos y Características

4.9. Lógicos o Booleanos

Unidad Didáctica II

Lógicos o Booleanos

CASO. Durante el desarrollo del programa de confección de nóminas, se recogen valores de datos del trabajador, entre los que se encuentran CASADO, TRANSPORTE y JORNADA CONTINUA, entre otros, datos que sólo tomarán dos valores "SI" o "NO", y que dependiendo de ese valor se aplicarán de una u otra forma al cálculo del salario del trabajador.

- ¿Cómo podríamos representar internamente esos dos únicos valores posibles?
- Por otra parte, ¿se te ocurre alguna operación posible con esos dos valores? Lo bueno de estos datos es que sólo existen dos posibilidades, por lo que se pueden almacenar usando únicamente un bit. (0 ó 1)

Todos los lenguajes incluyen un tipo de **datos lógicos o booleanos.Son datos que sólo pueden tomar dos valores distintos, (verdadero o falso, si o no, 0 ó 1)** Se utilizan normalmente para comprobar el valor de una condición, o una comparación de valores o para distinguir la ocurrencia de un suceso de entre dos posibles.

Para saber más:
Reciben el nombre de booleanos en honor del matemático inglés George Boole, que ideó una herramienta matemática denominada álgebra de Boole binaria, y que es la base matemática para la construcción de circuitos lógicos, y que permite aplicar la lógica matemática a la construcción de ordenadores. Para conocer algo más sobre Boole, sigue el siguiente enlace.
http://www-etsi2.ugr.es/alumnos/mlii/Boole.htm **[Versión en Caché]**

(Si tienes problemas para acceder a algún enlace, pulsa en "Versión en Caché" para visualizar una copia de esa página web)
Para descargar el programa Acrobat Reader pulsa aquí.

```
Los operadores habituales para este tipo de datos son
la conjunción lógica o Y-lógico (AND),
la disyunción lógica u O-lógico (OR) y
la negación lógica o NO-lógico (NOT).
```

La **tabla de verdad** de estos operadores, suponiendo a y b variables de tipo lógico, es la siguiente:

a	NOT a
V	F
F	V

a	b	a AND b	a OR b
V	V	V	V
V	F	F	V
F	V	F	V
F	F	F	F

Existen otros operadores lógicos, pero sólo los tres indicados anteriormente son imprescindibles, por lo que son los que están disponibles en todos los lenguajes.

```
En Java, el tipo lógico recibe el nombre de boolean,
y los valores posibles son: true (verdadero) y false (falso).
El operador ! es la negación lógica.
El operador || es la disyunción lógica.
El operador&& es la conjunción lógica.
```

Autoevaluación

Indica el valor de la tabla verdad de los siguientes operadores, suponiendo a y b variables de tipo lógico:

- a) a=verdadero or b=falso es VERDADERO.
- b) a=verdadero and b=verdadero es FALSO
- c) a=falso and b=verdadero es VERDADERO.
- d) a=falso or b=falso es VERDADERO

Comprobar

Datos: Tipos y Características

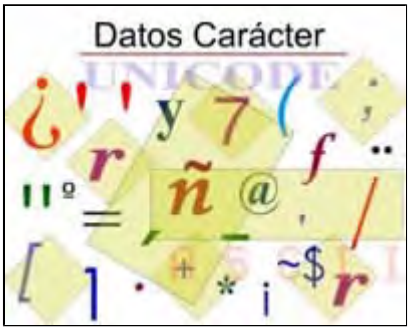
4.10. Carácter

Unidad Didáctica II

Carácter

CASO. En el programa que están desarrollando ahora nuestros amigos, hay una serie de datos que necesariamente deben ser definidos como cadenas de caracteres, como por ejemplo el NOMBRE y los APELLIDOS del trabajador, también hay algunos de tipo carácter como la LETRA DEL NIF, pero además deciden utilizar algunos otros también como caracteres, aunque en principio no parezca lógico, como por ejemplo el teléfono ya que al tratarse de números de nueve dígitos es mejor tratarlos como cadenas de caracteres que como datos reales.

Ya nos encontramos cerca de terminar el repaso a los tipos básicos que incluyen la mayoría de los lenguajes. Uno de esos últimos tipos son los datos de **tipo carácter.** Así que entremos en situación, que el tema avanza. ¿Qué nos representa el tipo carácter?



Los datos de tipo carácter representan elementos individuales del conjunto de caracteres usado como código de entrada-salida. El código más usual es el **ASCII**, que usa 8 bits (1 byte) para representar cada carácter, por lo que permite usar hasta 256 caracteres distintos, lo cual es bastante limitado a la hora de representar todos los posibles símbolos de todos los idiomas y alfabetos existentes. Aunque es el más extendido, no es el único posible. Por ejemplo, Java ha incorporado como alfabeto el código **Unicode**, más reciente que el código ASCII, y que usa 16 bits para representar cada carácter, por lo que puede representar hasta 65.536 caracteres distintos, lo que hace posible representar todos los alfabetos de todas las lenguas, y todo tipo de símbolos gráficos, matemáticos, caracteres de control, etc. Además, el código Unicode es "compatible" con el código ASCII, ya que para los caracteres del código ASCII, Unicode asigna como código los mismos 8 bits, a los que les añade a la izquierda otros 8 bits todos a cero. La conversión de un carácter ASCII a Unicode es inmediata.

No suele haber operadores asociados al tipo carácter, salvo algunas funciones que obtienen el código de un carácter concreto, o que dado un número obtienen el carácter cuyo código es ese número.

En Java, el tipo carácter se llama ***char***, y el alfabeto o código usado es Unicode. Tiene además la peculiaridad de que es considerado por el compilador como un tipo de entero, por lo que puede operarse con caracteres como si fuesen números enteros, ya que realmente se opera con su código Unicode, que es un número.

*Recuerda el ejemplo de la tabla hecha por **Carmen** a **Víctor**. Allí veíamos lo oportuno de usar el tipo char para el sexo de una persona, que tenía 3 posibilidades, identificadas por las letras 'H', 'M' y 'D'.*

Para saber más:
Para una lectura más detallada de los códigos de E/S puedes visitar la siguiente página:
http://www.conectados.org/doc/Apuntes/1_Elementos_de_Programacion/temai.pdf
[Versión en caché]

(Si tienes problemas para acceder a algún enlace, pulsa en "Versión en caché" para visualizar una copia de esa página web)
Para descargar el programa Acrobat Reader pulsa aquí.
De entre los distintos códigos de E/S, hemos destacado dos por su uso mayoritario, ASCII y Unicode, puedes saber algo más de ellos en los siguientes enlaces:

http://es.wikipedia.org/wiki/ASCII
http://www.unicode.org/

Datos: Tipos y Características

4.11. Fecha

Unidad Didáctica II

Fecha

CASO. En la aplicación que están desarrollando nuestros protagonistas, uno de los datos a almacenar sobre cada trabajador es su fecha de ingreso en la empresa. ***Víctor*** *ha pensado que lo puede almacenar como tres números enteros, uno correspondiente al día, otro al mes y otro al año. Eso sí, tiene que hacer un buen programa para que por ejemplo no admita días superiores a 31 para los meses.*
Carmen *le corrije: "según para qué mes, Febrero sólo tiene 28 días y hay varios meses que sólo tienen 30. Además deberás comprobar si el año es bisiesto cuando metas días para Febrero, porque en ese caso, 29 será un valor correcto. Además necesitarás hacer algunas funciones como ordenar un conjunto de fechas de más reciente a más antigua o viceversa, manipulando las tres variables, o cálculo de días entre dos fechas, o fecha que se obtiene al sumar un número de días a otra fecha, que serán necesarias en la aplicación para el establecimiento y gestión de turnos de trabajo, etc".*
Víctor *se da cuenta que no es tan sencillo como pensaba.*

Con el tipo fecha terminamos la presentación de los tipos básicos más usuales en gran parte de los lenguajes de programación. ¿Crees posible almacenar una **fecha** con los datos primitivos vistos hasta ahora?

Aunque no todos los lenguajes disponen de este tipo de datos, sí es frecuente que exista en muchos de ellos, ya que es frecuente que los programas trabajen con fechas (reservas de billetes, fechas de facturas, de entrega de pedidos, de nacimiento, de ingreso en la empresa, etc.) Y aunque podemos almacenar una fecha en tres variables de tipo entero, su gestión no resulta tan fácil como pueda parecer. Sobre todo si necesitamos más precisión en la medición del tiempo, y también queremos gestionar horas, minutos, segundos, décimas de segundo, milésimas de segundo, ... Por eso muchos lenguajes lo incluyen como un tipo básico, que se gestiona de forma automática por el compilador y cómoda para el programador, que se puede olvidar de los detalles. Se suelen introducir en un formulario con el formato dd/mm/aa, por ejemplo, al igual que solemos escribirlas. Será el compilador el que rechazará los valores que no sean correctos para el día del mes, o el que nos permitirá ordenar las fechas de menor a mayor (de más antigua a más reciente) Esto es así en todos los sistemas gestores de bases de datos, pero no en todos los lenguajes.

En Java no existe el tipo fecha como un tipo básico, aunque existen clases que definen toda una amplia gama de datos estructurados para la representación de fechas y operaciones posibles sobre fechas. Esas clases son fundamentalmente Calendar, GregorianCalendar y Date. Aparecerán más adelante en nuestros programas Java, y entonces será el momento de estudiarlas, ya que son extensas y tienen cierta complejidad.

A ***Víctor*** *le hubiera gustado que Java dispusiera del tipo básico fecha. Seguramente la aplicación se habría simplificado su trabajo con la aplicación para procesar las fechas de ingreso en la empresa de los trabajadores.* ***Carmen*** *le dice que por esta vez, se encargará ella de hacerlo usando las clases oportunas de Java, pero que más le vale ir tomando nota.*

Datos: Tipos y Características

5. Datos Estructurados

Unidad Didáctica II

Datos Estructurados

*CASO. Durante el diseño del programa para la elaboración de nóminas de trabajadores de la empresa **Servicios Locales**, **María**, que ha tenido que sustituir a **José** por un imprevisto, les explica a **Carmen** y **Víctor** las estructuras de datos que van a utilizar para recoger los datos de los empleados. **Víctor**, que ya ha tenido bastante con los tipos de datos, cree que esto supone un nuevo problema para comenzar con la programación. **María** le explica que es todo lo contrario. Que los datos estructurados van a facilitar mucho las cosas y que su utilización es mucho más intuitiva a nuestra forma de pensar que cualquier otra cosa, ya que de hecho han sido definidos para mostrar los datos de forma agrupada y con una estructura más acorde con nuestra manera de verlos.*

¿Se te ocurre algún tipo de **dato** de la vida real que sea difícil representar y procesar usando únicamente los tipos de datos básicos estudiados hasta ahora? Seguramente más de uno. Un empleado de la empresa es algo más que una serie de datos individuales, podemos verlo como una estructura que aglutina todos esos datos de tipo básico. Otro **ejemplo** podría ser una serie de valores que represente, con fines estadísticos, la altura en cm de una muestra de 5.000 personas, sobre los que tenemos que calcular variables estadísticas (media, moda, mediana, varianza, desviación típica, etc.) Usar 5.000 variables de tipo entero sería inmanejable. ¡Imagínate escribiendo la suma de las 5.000 variables en una expresión, cada una con su nombre! Es por eso que necesitamos **estructuras de datos**, que nos faciliten la tarea de **gestionar** gran cantidad de datos, o que sencillamente nos permitan representar los datos reales de una forma más fiel.

Además de los tipos simples, o básicos o primitivos, **la mayoría de los lenguajes permiten usar una serie de estructuras de datos algo más complejas, que se construyen a partir de otros tipos, que pueden ser los propios tipos básicos o primitivos o ser a su vez tipos estructurados, o tipos definidos por el usuario**. En este apartado se han incluido los tipos estructurados que suelen proporcionar ya definidos los lenguajes, aunque el usuario puede definir también sus propias estructuras de datos a través de la definición de objetos, que son estructuras y podrían incluirse aquí.

Gran parte del aprendizaje de un lenguaje consiste en saber usar sus tipos de datos tanto simples como estructurados, por lo que aquí sólo indicaremos brevemente en qué consisten estas estructuras, dejando una explicación más completa de su uso y funcionamiento para las unidades 8 y 9, justo antes de empezar a introducir el lenguaje Java.

__Víctor__ tiene dudas. Le acaban de pasar un documento con toda la información que debe almacenar para cada empleado de la empresa, y éstos son muchos. Le dice a __Carmen__ que si para cada uno va a tener que usar variables distintas, no va a terminar el programa ni para el día del juicio. ¡Sólo para guardar el nombre de cada empleado necesitaría tantas variables de tipo cadena de caracteres como empleados! __Carmen__ le dice que no diga disparates. Habrá que usar alguna estructura de datos que simplifique el manejo de toda esa información. __Carmen__ responde que aún no lo sabe porque __José__ no ha terminado de hacer el análisis de la aplicación, pero a simple vista, parece que una posibilidad sería crear una __clase__ Empleado, que permitiera crear __objetos__ de tipo Empleado, con todos sus datos, y gestionarlos en memoria mediante una __lista enlazada__ de empleados, que podría estar ordenada alfabéticamente, y que podría guardarse en un __fichero__ en disco para tener todos los datos a salvo. Aunque si el número de empleados es realmente alto, quizás una __Base de Datos__ sería la solución más oportuna.

Datos: Tipos y Características

5.1. Cadenas de caracteres

Unidad Didáctica II

Cadenas de caracteres

*CASO. Las **cadenas** son probablemente las estructuras de datos más utilizadas, ya que se emplean para recoger casi cualquier tipo de dato. Como dice **Jesús** a sus compañeros; "todo son cadenas salvo los valores numéricos necesarios para cálculos y las fechas". De hecho tiene razón, ya que podemos hablar de cadenas de caracteres como el dato de tipo general para recoger cualquier valor, cuyas excepciones serán los valores numéricos (empleados en operaciones aritméticas), las fechas (que tienen un tratamiento especial), los datos lógicos y algún dato que sólo necesite un carácter.*

En general, **una cadena de caracteres es una sucesión de caracteres (letras, números y demás caracteres del alfabeto del lenguaje)**. Se utilizan normalmente como un tipo de dato predefinido, para palabras, frases o cualquier otra sucesión de caracteres. Un **ejemplo** de valor posible para una cadena de caracteres es:

- el nombre de una persona,
- una frase de un libro,
- la contraseña de un usuario que accede a un curso a través de internet, o
- el número de teléfono,

que aunque no contenga más que dígitos entre sus caracteres, podemos almacenarlo como de tipo cadena de caracteres, ya que no vamos a realizar ningún tipo de operación matemática con él.

Es muy habitual que se delimiten mediante comillas (") al principio y al final de la cadena. Como un **ejemplo** concreto podemos ver la cadena de caracteres:

```
"Esto es una cadena de caracteres"
```

Para poder mostrar, por ejemplo, una comilla (") dentro de la cadena y no tener problemas con las comillas que la delimitan, se usan **secuencias de escape**. Esto se aplica a otros caracteres reservados o no imprimibles como el retorno de carro o salto de línea. No obstante, las expresiones para producir estas secuencias de escape dependen del lenguaje de programación que se esté usando.

Una forma común, en muchos lenguajes y en Java en concreto, de "escapar" un carácter es anteponiéndole un «\» (sin comillas). Por ejemplo:

```
"la cadena \"ejemplo\" contiene comillas "
```

realmente será interpretada por el lenguajes como la cadena:

```
la cadena "ejemplo" contiene comillas
```

Algunas operaciones comunes con cadenas son:

- **Concatenar dos cadenas de caracteres para formar una nueva**.

En Java, la concatenación de cadenas se consigue con el operador +. Con un ejemplo, quizás es más claro:

```
"El empleado encargado de la sección es "+"José García" genera la cadena:
El empleado  encargado de la sección es José García
```

- **Obtener una subcadena a partir de otra**. Un ejemplo con Java sería

```
"El empleado encargado".substring(4,12)  devuelve la subcadena empleado
```

- **Obtener el carácter de una posición de la cadena**. El ejemplo Java sería:

```
"Empleado".charAt(2)  devuelve el carácter de la posición 3,
 que es 'p' (La primera posición es la cero)
```

- **Buscar una subcadena o un carácter dentro de la cadena, y saber en qué posición se encuentra**.
- **Calcular la longitud de una cadena**

```
"Empleado".legth() devuelve el valor 8
```

- **Comprobar si empieza o termina con una secuencia de caracteres**

```
"Empleado".startsWith("Em") devuelve true (verdadero).
"Empleado".endsWith("a")devuelve false (falso)
```

- **Reemplazar parte de una cadena por otra**

```
"El empleado  encargado".replaceAll("e","X") cambia todas las 'e' por 'X',
quedando la cadena como El XmplXado  Encargado.
Fíjate que sin embargo la letra 'E' no cambia, ya que para el lenguaje se trata
de un carácter totalmente independiente a la letra 'e'.
```

En Java, las cadenas de caracteres se forman como objetos de las clases ***String***, o ***StringBuffer***, que ya vienen incluidas como parte del kit de desarrollo Java (JDK), pero que pueden considerarse como "extensiones" al núcleo del lenguaje.

Datos: Tipos y Características

5.2. Vectores y Matrices (Arrays)

Unidad Didáctica II

Vectores y Matrices (Arrays)

*CASO. Nuevamente **Víctor** se siente desplazado cuando empiezan a hablar de **Arrays**, ya que nunca ha oído hablar de esto. Además se ve sorprendido dada la importancia que todos le dan a este concepto y el uso que le están dando desde que entró en la empresa. Ahora es el momento en que se ha encontrado de frente con los arrays y ya no puede esquivarlos más. Debe enfrentarse a esta **estructura de datos**, y de hecho tiene ganas de enterarse de una vez por todas en qué consiste y por qué son tan importantes.*
***José** le ha encargado que haga un método que mantenga un array de cadebas de caracteres (String en Java) llamado puestosDeTrabajo con la lista delas descripciones de los puestos de trabajo existentes en la empresa. Deberá añadirle nuevos puestos de trabajo que se creen, o eliminar los que hayan dejado de existir.*
*Además deberá hacer otro método boolean asignaPuestoTrabajo(String puesto), que debe usarse para asignar o modificar el puesto de trabajo de un empleado. El método sólo asignará ese puesto de trabajo si aparece en el array de puestos de trabajo de la empresa. Si es un puesto correcto, el método lo asignará y devolverá el valor verdadero para indicar que la asignación se hizo con éxito. En caso contrario, devolverá el valor falso, sin asignar ningún valor al campo puesto de trabajo del emplado. Víctor está tranquilo. Sabe que **Carmen** de eso sabe un montón, y que le va a enseñar qué son los arrays.*

Un *array* es un conjunto de variables o registros del mismo tipo que puede estar almacenado en memoria principal o en memoria auxiliar.

- Los arrays de 1 dimensión se denominan ***vectores***,
- los de 2 o más dimensiones se denominan ***matrices***.

En definitiva **un array es un conjunto de posiciones consecutivas de memoria, en la que se guardan datos del mismo tipo, y que recibe un único nombre, el nombre del array:**

- **Cada posición guarda un dato individual,**
- **tiene un número asociado (un índice) que indica el lugar que ocupa dentro del array**
- **Para acceder directamente a un dato individual (para darle valor, modificarlo o consultarlo...) lo hacemos mediante el nombre del array seguido del índice.**

El nombre del **array** localiza la dirección de comienzo en memoria de la estructura, mientras que el índice representa el desplazamiento respecto a esa dirección de comienzo para llegar al elemento deseado.

Al declarar un array, deberemos decir cuantas **dimensiones** tiene, y cuantos **elementos** en cada dimensión. Algunos lenguajes obligan a hacer esto durante la compilación del programa (en tiempo de compilación), y otros permiten hacerlo durante la ejecución del programa (en tiempo de ejecución). En lo que suelen coincidir es en no limitar el número de dimensiones posibles. A nosotros nos cuesta imaginar más de tres dimensiones, porque lo asociamos al espacio, pero conceptualmente, no hay por qué limitarlo.

Por **ejemplo**, podemos representar las distintas mesas o puestos de trabajo de una empresa, de forma que:

1. La **primera** dimensión represente el edificio concreto de la empresa.
2. Dentro de cada edificio hay varias plantas, representadas por la **segunda** dimensión.
3. Dentro de cada planta, hay varios pasillos, representados por la **tercera** dimensión.
4. Dentro de cada pasillo, hay distintos despachos, representados por al **cuarta** dimensión, y
5. dentro de cada despacho hay varias mesas, representadas por la **quinta** dimensión.

Cuando hablamos de matrices, vectores y arrays, tendemos a limitar el concepto de array al de tabla, pero en informática el concepto de array o tabla es mucho más amplio.

En Java, para **definir un array**, se pone el tipo de los elementos individuales, seguido de un corchete por cada dimensión que queramos darle, y del nombre del array. Posteriormente habrá que dimensionarlo, indicando cuantos elementos va a tener, y posteriormente se usará para almacenar valores con algún propósito. Operaciones típicas con arrays son ***recorrerlo***procesando todos sus elementos (llenarlo, etc.), ***consultar*** el valor de un elemento, ***modificar*** un elemento (borrarlo, darle valor, actualizarlo)

Ejemplo en Java: (sólo para que vaya sonando)

```
...
int [ ][ ] arrayBidimensionalDeEnteros;
arrayBidimensionalDeEnteros = new int[3][4];
arrayBidimensionalDeEnteros[0][0]=23;
...
arrayBidimensionalDeEnteros[2][3]=376;
```

DEMO: En ese ejemplo se declara un array bidimensional de números enteros. A continuación decimos que va a tener 3 filas y 4 columnas, es decir, decimos cuantos elementos va a tener cada dimensión. Y finalmente le damos valor a algunos elementos individuales del array.

Datos: Tipos y Características

5.3. Registros

Unidad Didáctica II

Registros

*CASO. **José** dice a sus compañeros que los **registros** son probablemente la estructura de datos más completa que hay y que es muy utilizada a la hora de registrar datos referentes a individuos, productos o empresas, ya que facilitan considerablemente su recogida y manipulación. Y añade que cuando sean capaces de diseñar registros correctamente, tendrán a mano la posibilidad de guardar cualquier cosa en un ordenador.*

¿Para qué casos será apropiado usar registros?¿Son los registros algo parecido a las fichas de cartulina donde recogen los datos de cada trabajador actualmente en la empresa? Efectivamente, ésa es la idea: llevar a formato digital esa ficha en cartulina, de forma que corregir los datos de una ficha, mantenerlas ordenadas, crear una nueva ficha, o consultar los datos de una de las fichas correspondiente a un trabajador concreto sea más fácil.

Un registro es una estructura de datos formada por yuxtaposición de elementos de distinto tipo que contiene información relativa a un mismo ente u objeto. A cada uno de los elementos que componen el registro se les llama ***campos.*** Los campos aparecen en un orden determinado y se identifican por un nombre. **Para definir el registro, hay que darle un nombre y un tipo a cada campo**. **El tipo de cada campo puede ser una estructura de datos a su vez.**

Por **ejemplo**, para representar la información de un alumno de este módulo profesional, que requiere asignarle una nota para el trabajo final de cada una de las 21 unidades de que consta, se puede usar un registro ***alumnoPLE*** que contenga los siguientes campos: ***apellidos*** (cadena de caracteres), ***nombre*** (cadena de caracteres), ***edad*** (entero), ***sexo*** (carácter), ***repetidor*** (lógico) ***notasTrabajosTemas*** (vector de números reales, de 21 elementos)

Posteriormente podré crear variables de tipo alumnoPLE que contendrán los datos de un alumno concreto.

alumnoPLE

DNI	NOMBRE	APELLIDOS	FOTO

CORREO ELECTRONICO	TELEFONOS

POBLACION	DIRECCION

EDAD	SI TRABAJA, INDIQUE DONDE...	IDIOMAS

REPETIDOR	NOTAS TRABAJOS Y TEMAS

No existen operaciones fijas a realizar con un registro, pero las habituales son:

- consultar sus valores,
- modificarlos y actualizarlos,
- crearlos y
- borrarlos.

En Java no existen los registros como tales, pero se puede crear este tipo de estructuras como

objetos de una **clase**. En la clase definiré la estructura del objeto, indicando los campos de que consta, y el tipo de cada uno.

Víctor *parece tenerlo claro. Para almacenar los datos de los empleados de la empresa, necesitan almacenarlos en registros, uno por cada empleado.* ***Carmen*** *le dice que en Java tendrán que crear una clase Empleado, e ir creando objetos de esa clase para cada nuevo trabajador de la empresa. Le explica el ejemplo de la ficha de clase de cada alumno en el Instituto cuando estaban estudiando.*

Datos: Tipos y Características

5.4. Archivos o Ficheros

Unidad Didáctica II

Archivos o Ficheros

*CASO. Al igual que ocurre con los ficheros tradicionales en forma de grandes cajoneras que se emplean en las oficinas, los archivos o ficheros informáticos deben ser ordenados convenientemente para localizar de forma rápida y eficiente cualquier dato que hayamos guardado. **José** pide que todo el trabajo esté organizado en carpetas para que puedan localizar cualquier documento rápidamente y le dice a **Víctor**; "las cosas las guardamos porque serán útiles más adelante, si no fuese así ¿por qué querríamos guardarlas?".*

¿Pero cada registro es como una variable? ¿Si tengo miles de empleados tendré que gestionar miles de variables registro? ¿Si apago el ordenador perderé los datos de todos los registros que había creado en memoria? Claro que no. Para eso existen justamente otras estructuras de datos llamadas archivos o ficheros.

Un archivo es un conjunto de información sobre un mismo tema, tratada como unidad de almacenamiento y organizada de forma estructurada para la búsqueda y recuperación de un dato individual. Es por tanto la estructura que necesitamos usar para poder guardar los datos e informaciones en soportes de almacenamiento masivo o permanente, tales como discos duros, disquetes, CD's, memorias Flash, cintas magnéticas, etc. y poder recuperarlos cada vez que quiera usarlos en una aplicación sin tener que volver a introducirlos de nuevo. Si sólo se guardan en la memoria principal del ordenador, se perderán cuando éste se apague.

Habitualmente los **ficheros o archivos** están compuestos por una colección de registros homogéneos. Así, por **ejemplo** lo lógico es que los datos de los alumnos de este módulo profesional se guardaran en memoria permanente, en el disco duro de algún ordenador. Para ello lo normal es que se cree un fichero de alumnos que sería una colección de registros alumnoPLE, como los definidos en el apartado anterior.

Las operaciones típicas a realizar con un fichero son:

- Creación del fichero.
- Inserción de un registro
- Eliminación o borrado de un registro
- Consulta de uno, varios o todos los registros.
- Modificación o actualización de un registro.
- Borrado del fichero.
- Búsqueda y localización de un registro concreto (normalmente habrá que hacerlo antes de cualquier operación de recuperación o actualización de un registro.)
- Duplicado o copia de seguridad del fichero.
- Ordenación del fichero.
- Mezcla de los datos de varios ficheros para formar uno nuevo.

Muchas de las tareas de manipulación de los ficheros las realizará el sistema operativo y los programas que hagamos, normalmente realizarán peticiones al sistema operativo para gestionar los archivos. Eso permite que nuestro programa no tenga que entrar en los detalles concretos de los soportes de almacenamiento del ordenador en el que se ejecuta. De eso se encargará el Sistema Operativo por lo que el programa funcionará con cualquier dispositivo y en cualquier ordenador.

*Para **Víctor** ha quedado claro que la información de los empleados de la empresa estará estructurada en registros, y que esos registros deberán guardarse en ficheros de forma permanente. La empresa necesita sustituir el armario archivador con las fichas de cartulina de los empleados por un fichero de registros tipo "empleado"en el ordenador.*

Datos: Tipos y Características

5.5. Punteros

Unidad Didáctica II

Punteros

*CASO. Los punteros requieren un estudio aparte, ya que su ámbito de uso abarca desde los enteros a los ficheros, pasando por estructuras y objetos. No obstante, como dice **María**, de momento conviene saber que se trata de que podemos trabajar directamente con direcciones de memoria, algo muy útil cuando se trata de comunicar aplicaciones.*

Puede que a lo largo de este tema te hayas planteado la necesidad de gestionar alguna estructura sin conocer el tipo de elementos que va a almacenar:

- ¿Cambia en algo el algoritmo de ordenación de un vector por el hecho de que sus elementos sean números enteros en vez de números reales o en vez de nombres de personas?
- ¿Cómo escribirías un código para ordenar un vector que no puedes crear porque no sabes de qué tipos son sus elementos?
- ¿Tendrías que escribir versiones distintas del programa de ordenación para cada dato posible?

Este es uno de los problemas que podemos solucionar gracias a los **punteros**.

En programación **por puntero entendemos un tipo de dato que corresponde a una dirección de memoria que a su vez referencia a un dato de otro tipo.Una variable de tipo puntero lo único que va a contener por tanto es una dirección de memoria (una referencia) que será la que realmente contendrá el dato.** Al definir el puntero, en esa variable sólo se podrán guardar direcciones de memoria que contengan un dato del tipo especificado para el puntero. Por **ejemplo**, una variable de tipo "puntero a alumnoPLE" sólo podrá contener una determinada dirección de memoria si en ella hay almacenado un registro de tipo alumnoPLE.

Mediante los punteros se consigue usar y **gestionar** la memoria de **forma dinámica**, y generalizar algunos algorimos. Por **ejemplo**, conseguimos crear un array de punteros, y definir un algoritmo que ordene un array de punteros. En un momento dado, esos punteros apuntarán a números enteros o a números reales, o a nombres de personas. El algoritmo en sí no cambia. Ordenar, se ordena siempre de la misma manera, a base de comparaciones.

Cuando un **compilador** traduce un programa a código máquina necesita asociarle, antes de ejecutarlo, a cada nombre de variable una dirección de memoria, que se reserva para esa variable. Si sabemos el tamaño que va a ocupar el dato, esa dirección de memoria reservada puede ser de un tamaño suficiente para que contenga el dato, y de hecho contenerlo directamente. Esto es un **uso estático de la memoria**. Mientras el programa está funcionando tiene reservado ese trozo de memoria sólo para esa variable.

Sin embargo, hay muchos casos en los que no sabemos el **tamaño** que va a necesitar una determinada estructura de datos hasta que se está ejecutando el programa, (por **ejemplo**, no sabemos lo larga que va a ser una cadena de caracteres que va a recibir valor desde teclado).

No podemos reservar previamente la memoria porque **desconocemos** el tamaño (puede ser desde una sola letra hasta un capítulo de un libro). Sólo se podría hacer reservando siempre un **tamaño máximo**, que sería el que siempre se reservaría. Pero eso tendría el inconveniente de que desperdiciaría mucho espacio de memoria para cadenas pequeñas o limitaría el tamaño innecesariamente.

reservar sólo el espacio de memoria que ocupa una dirección de memoria. Cuando el programa ya esté ejecutándose y se sepa el tamaño exacto de esa cadena, buscaremos un trozo libre de memoria donde alojarla, y anotaremos en nuestra variable de tipo puntero la dirección de memoria donde comienza la cadena. Siempre que en el programa se use la variable puntero, accederemos a la misma posición de memoria, en la que nos encontraremos la dirección de memoria en la que estará el dato. Si en un momento dado esa cadena cambiase de tamaño y necesitase más espacio, bastaría con buscar nueva memoria libre, alojarla, liberar la memoria que antes ocupaba para que pueda ser usada por otras variables, y anotar en el puntero la nueva dirección de memoria donde comienza la cadena. Con esto se consigue una **gestión dinámica de la memoria**.

Además los punteros son imprescindibles para crear otras estructuras de datos "enlazadas", tales como listas enlazadas o árboles, que se verán más adelante.

En **Java** no existen punteros como tales, pero sí existen **referencias**, que conceptualmente son casi lo mismo, pero de manejo y gestión mucho más simple. Ello se debe básicamente a que una vez que reubicamos el valor de una variable, sólo nos tenemos que preocupar de actualizar la referencia. Hay un programa llamado recolector de basura que se encarga de liberar sin nuestra intervención la memoria que ya ha dejado de usarse.

Víctor *piensa que quizás necesiten hacer una lista enlazada de empleados en la memoria con los datos que se lean del fichero, para poder manejarlos más cómodamente en el programa. Va a necesitar repasarse los punteros. ¿Es complicado el uso de punteros? Un poco, le responde* ***Carmen****. Pero alégrate. En Java no hay punteros, sino referencias, que son mucho más fáciles de gestionar y de entender que los punteros de otros lenguajes. Aunque en el fondo son lo mismo.*

Autoevaluación

Señala la afirmación correcta:

- a) Mediante los punteros se consigue usar y gestionar la memoria de forma estática.
- b) El motivo de usar punteros es que desconocemos a priori el tamaño que va a necesitar una determinada estructura de datos.
- c) Las variables de tipo puntero almacenan el dato que vamos a utilizar en un programa.
- d) Un inconveniente del uso de variables de tipo puntero es que como son variables dinámicas de memoria, los valores de dichas

variables desaparecen al apagar el ordenador (se borra la memoria principal).

Comprobar

Datos: Tipos y Características

5.6. Listas

Unidad Didáctica II

Listas

*CASO. Cuando escucha hablar a sus compañeros de **listas enlazadas**, **Victor** no termina de entender a qué se refieren, cree que ya no puede más con tanta estructura de datos y le parece increíble que sus compañeros las utilicen todas en una sencilla aplicación que él habría programado en dos semanas. Aunque reconoce que las aplicaciones obtenidas por el grupo funcionan, son más eficientes y son infinitamente más fáciles de mantener. **Carmen** intenta explicarle que las listas son una serie de punteros enlazados entre sí. Incluso le ha hecho un dibujo, pero él no termina de ver clara su utilidad. Finalmente **Carmen** decide no dedicarle más tiempo y le dice que lo mejor es que programe una, por ejemplo haciendo una simulación de la cola de una ventanilla en la que no está permitido colarse.*

Usar ficheros para almacenar la información está muy bien, pero ¿puedo yo **trabajar directamente** sobre el fichero almacenado en disco? Depende de lo que se entienda por directamente. Lo habitual es que no. Los datos almacenados en el fichero se leerán y guardarán provisionalmente en algunas variables o estructuras de datos en memoria, para gestionar la información. Una posibilidad sería leer todos los registros del fichero y guardarlos en una lista enlazada en memoria para trabajar sobre ella.

Como ya hemos dicho antes, gracias a los punteros puede gestionarse la memoria de forma dinámica, y ello permite crear estructuras de datos dinámicas. **Un tipo de estructura dinámica son las *Listas enlazadas*, que pueden incluir muchas variantes, y que se estudiarán con detalle, así como su uso y funcionamiento en un**a unidad posterior.

Por ahora basta con decir que una lista enlazada está compuesta por:

- **Distintos elementos o nodos**, cada uno de los cuales será un registro del mismo tipo, dispuestos de forma secuencial.
- **En cada nodo se incluye un campo que sea de tipo "puntero al siguiente nodo de la lista**"
- **Un puntero de inicio que siempre apunte al primer elemento de la lista**.
- **El campo "puntero al siguiente nodo de la lista" del último nodo debe apuntar a un valor especial**, que normalmente recibe el nombre de ***Null***, y que indica que no hay ningún otro elemento detrás.

Por **ejemplo**, en el caso de que queramos hacer una lista de empleados, necesitaremos punteros de tipo empleado. Un puntero apuntará siempre al primer empleado de la lista.

Cada nodo de la lista será un registro empleado, al que además de la información del empleado habremos añadido un puntero de tipo empleado, al que llamamos "puntero al siguiente empleado".

El puntero al siguiente empleado apuntará la siguiente empleado (contendrá la dirección de memoria donde está el registro del siguiente empleado).

El último empleado de la lista tendrá un valor Null , para indicar que no hay más registros de empleado en la lista.

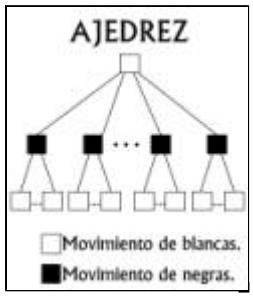
Datos: Tipos y Características

5.7. Árboles.

Unidad Didáctica II

Árboles

CASO. Después de vérselas con las listas, ***Victor*** *dice que los árboles sí que los conoce,sobre todo porque ha estado viendo el funcionamiento de un programa de ajedrez y en la revista ponía que la programación de todas y cada una de las posibilidades de movimientos se hacía con una estructura de árbol en el que la profundidad del mismo determinaba la probabilidad de acorralar al rival.*



Hemos visto las listas enlazadas en el apartado anterior, y si no has tenido problemas para entender en qué consisten, tampoco los tendrás con los árboles. ¿Qué diferencias hay entre árboles y listas enlazadas, entonces? Al igual que las listas, **un árbol es una estructura de datos dinámica, que se construye usando punteros**. La diferencia es que se trata de una **estructura jerárquica**.

En un árbol tendremos:

Un nodo llamado raíz del árbol por el que se accede a la estructura.

- **Cada nodo puede tener varios nodos hijos**, cada uno de los cuales se puede considerar como la raíz de un nuevo subárbol, y que constituye una rama del árbol general.
- **Cada nodo deberá contener campos de tipo "puntero a nodo hijo"** para cada uno de los posibles hijos.
- **Para un nodo concreto, sólo existe un nodo padre**, por lo que un nodo no puede pertenecer más que a una rama del árbol.
- **Cuando una rama se termina, los campos que apuntan a los hijos, deben contener el valor Null**.
- **Es necesario disponer de un puntero que apunte siempre al nodo raíz del árbol**, para poder acceder a la estructura, ya que si no sería imposible usarla, aunque estuviera en memoria ocupando espacio.

Se estudiarán los tipos de árboles su funcionamiento y utilidad con más detalle en próximas unidades.

Unidad Didáctica II

Datos Definidos por el usuario

*CASO. Además de los datos estructurados que hemos visto, a un personaje como **Víctor** ya sólo le faltaba el que un programador tuviera la posibilidad de crear nuevos tipos adaptados a sus necesidades. Cuando **María** le explica que a veces es interesante utilizar subtipos que ellos mismos definen según las necesidades del cliente. **Víctor** ya no entiende nada y piensa que le están tomando el pelo.*

Algunos lenguajes permiten **definir** datos al **usuario** mediante distintos mecanismos. De ellos, los más útiles son los **Objetos**, en aquellos lenguajes que permiten programación orientada a objetos, ya que sí permiten definir auténticas estructuras de datos, tan complejas como queramos, en cuanto a valores posibles como a operaciones permitidas. Los otros tipos que se mencionan (enumerados y subrangos) son menos útiles, y hay muchos lenguajes que ni siquiera disponen de mecanismos para definirlos o usarlos.

6.1. Enumerados

Unidad Didáctica II

Enumerados

*CASO. Para convencer a **Víctor** de la utilidad de este tipo de datos, **Jesús** le explica que es sólo una posibilidad que en un momento dado puede facilitar las cosas. Que es posible que no los utilice y que probablemente sea posible programar prescindiendo de estos tipos. Pero le pone el siguiente ejemplo para que compruebe por sí mismo que en ocasiones pueden ser interesantes. Un tipo Enumerado permite definir los valores que puede tomar un semáforo, con sólo VERDE, AMARILLO y ROJO. Si se define el tipo SEMÁFORO de tipo Enumerado, no podrá presentar un estado diferente al de esos tres colores, es decir, cualquier variable de tipo Semáforo sólo va a poder tomar uno de esos tres colores.*

Piensa en la cantidad de veces que aunque los datos posibles sean de tipo cadena de caracteres, no todas las cadenas de caracteres serán válidas. Por **ejemplo**, cualquier cadena de caracteres no es válida como día de la semana. El tipo enumerado a usar estaría compuesto por Lunes, Martes, Miércoles, Jueves, Viernes... Estaríamos definiendo el tipo por enumeración de los elementos que son datos de ese tipo. Estos tipos no pueden tener mucha operatoria asociada, más que la de comprobar si un elemento pertenece al tipo o no, o añadir y borrar elementos al tipo.

Su utilidad también es limitada.

En Java existe algo parecido a través del **interface** ***Collection***, y de toda una serie de clases que implementan ese interfaz (28 en total. Es pronto para entrar en detalles de explicarlas todas. Pero todas definen **colecciones** de objetos, en abstracto. En cada caso, la colección estará formada por objetos concretos, pero ni siquiera es obligatorio que sean del mismo tipo. Podemos tener en la misma colección un empleado, un animal, un cohete y un número entero. La colección nos permite preguntar si un determinado objeto pertenece o no a la colección.

En otros lenguajes, como Pascal, sí existía el tipo set, que es un tipo enumerado (tipo conjunto) y que lo único que tenías que hacer era que indicar el tipo de los elementos. Podían ser los que quisieras, pero todos del mismo tipo.

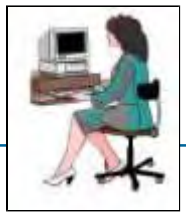
Datos: Tipos y Características

6.2. Subrango

Unidad Didáctica II

Subrango

*CASO. **Carmen** explica que éste es un tipo poco usado y básicamente se trata de seleccionar un conjunto de valores válidos para un dato concreto. Por ejemplo podemos definir un tipo subrango para referirnos a las calificaciones de los alumnos (enteros positivos de 0 a 10).*



Algo similar a lo que ocurría con los días de la semana nos ocurre con los días del mes, por ejemplo. Son valores válidos para el día del mes los números enteros positivos comprendidos entre 1 y 31, lo que supone un **subrango** del rango de números enteros. Y eso es lo que queremos poder definir, un tipo como un subrango de un tipo ya definido, siempre que ese tipo tenga establecida una relación de orden entre los elementos que lo forman. Consiste en indicar que el nuevo tipo sólo va a admitir valores desde uno inicial hasta otro final de los del tipo padre.

Así, por **ejemplo**, podríamos definir el tipo DiaDelMes como un subrango de los enteros que comienza en el 1 y termina en el 31.

Como dijimos antes, su utilidad es muy limitada.

***Víctor** se queda pensativo. ¿Para el problema del método que debe analizar si un puesto de trabajo está en la lista o no, podría usar algún tipo enumerado o subrango que estableciera cuales son los puestos de trabajo válidos? Sería una alternativa, le contesta **José**. Pero en Java viene a ser lo mismo que hacer tú el vector con los valores permitidos.*

6.3. Objetos

Unidad Didáctica II

Objetos

*CASO. **Carmen** recuerda cómo en sus tiempos de estudiante éste era uno de los conceptos que más le costó entender, ya que parecía que en principio todo podía ser un **objeto**, pero uno de sus profesores le aclaró la confusión comentándole que los objetos deben cumplir ciertos cánones. Se definen de una determinada forma y eso le aporta una serie de propiedades, de modo que otros del mismo tipo van a heredar algunas de sus características y tendrán un comportamiento similar ante determinadas operaciones.*

Decir que algo es un objeto viene a ser como decir que has usado un cacharro. Es una definición bastante ambigua, en principio. Pero eso es lo que deliberadamente queremos en el caso de los objetos. ¿Qué es un objeto? ¿Qué nos aporta?¿Para qué sirve?

Los **objetos** tratan de permitir una libertad total al definir un tipo de dato:

- Nos permiten definir cualquier estructura de datos,
- cualquier conjunto de valores posibles,
- cualquier conjunto de operaciones que se pueden realizar con esos datos
- cualquier grado de libertad en la definición que permita adaptar los programas a las cosas del mundo real.

En la vida todo son **objetos**, y lo que tratamos es de que el programador pueda hacer un modelo de tipo de datos a la medida del objeto de la vida real que quiere representar.

Realmente se trata de un tipo estructurado y definido por el usuario. Está disponible para los lenguajes orientados a objetos, que son la mayoría de los que se usan actualmente.

Nos proporcionan un mecanismo realmente potente y flexible de definir cualquier tipo de datos que deseemos, o cualquier estructura de datos que podamos imaginar.

A través de la definición de una clase, se define una estructura de datos, tanto en lo relativo a qué valores va a tomar como en lo relativo a qué operaciones se van a poder aplicar a esos valores.

En la clase se puede definir la estructura del objeto de forma similar a como se definía un registro, **indicando qué campos (variables miembro de objeto) va a tener, y de qué tipo van a ser, junto a otras características que van a limitar quien va a tener acceso a cada campo, y qué tipo de acceso.** Naturalmente, cada variable que forme parte del objeto puede ser de cualquier tipo estructurado, incluyendo cualquier otra clase ya definida, o incluso la propia clase (similar a los punteros de las listas enlazadas)

Por otro lado, en la clase podemos definir todo un conjunto de métodos (o procedimientos) en los que se define qué uso se va a poder hacer de esos datos,

junto a restricciones de acceso y seguridad.

Y toda esa estructura se puede crear como una auténtica caja negra, de la que conozco qué le puedo dar y qué puedo esperar, pero nada acerca de cómo se realizan los procesos internamente. **A esto se le llama ocultación de la información**.

Datos: Tipos y Características

6.4. Características que definen los objetos a través de un ejemplo

Unidad Didáctica II

Características que definen los objetos a través de un ejemplo

Los objetos serán variables del tipo que define la clase, aunque en la terminología de la programación orientada a objetos se les suele llamar instancias de la clase.

Por ejemplo, yo puedo definir la clase AlumnoPLE, en la que indico:

- **los campos que va a tener un objeto de tipo AlumnoPLE**:
  - **apellidos** (cadena de caracteres)
  - **nombre** (cadena de caracteres)
  - **edad** (entero)
  - **sexo** (carácter)
  - **repetidor** (lógico)
  - **notasTrabajosTemas** (vector de números reales, de 21 elementos)

Con este apartado estoy definiendo **la estructura de los datos de este tipo**.

- **Las operaciones a realizar con un alumno:**
  - Crearlo
  - Ponerle nombre
  - Ponerle apellidos
  - Rellenar cualquier otro campo
  - Actualizar los valores
  - Imprimirlo
  - ponerle nota en el trabajo de un tema
  - calcularle la nota media si ya tiene nota en todos los trabajos.

Con este último apartado estoy definiendo también **el comportamiento de los objetos que sean del tipo AlumnoPLE**.

Cada variable que declare como de tipo AlumnoPLE va a ser una nueva instancia de la clase, o lo que es lo mismo, un nuevo objeto de esa clase.

Además esto aporta otra serie de beneficios, como la facilidad para reutilizar código, que se verán con más detalle en las unidades correspondientes a programación orientada a objetos.

*El ejemplo de los alumnmos de PLE es fácilmente trasladable a los empleados de la empresa, por lo que cada vez **Víctor** ve más claro que debe aprender a sacarle partido a esto de las clases y los objetos si quiere ser un auténtico programador.*

7. Bases de Datos

Unidad Didáctica II

Bases de Datos

*CASO. **María** dice que después de dar tantas vueltas, lo que va a utilizar la empresa para recoger los datos de sus trabajadores, va a ser una Base de Datos. Evidentemente **Victor** le pregunta que entonces qué sentido tiene conocer otras estructuras de datos si al final todo el mundo usa las bases de datos, que él lo ha leído en la última revista de informática para programadores. **María** le contesta con muy buen criterio que las bases de datos son unas estructuras complejas, formadas por registros, ficheros, punteros, etc. Una vez más **Victor** tiene que darse por vencido y aplacar su impulso que continuamente le hace cuestionar cualquier decisión de sus compañeros, aunque parece que está empezando a admitir que algunas cosas existen porque son útiles, especialmente en el mundo de la programación de ordenadores.*

Seguramente sabes al menos de forma aproximada lo que es una **base de datos**. Incluso serías capaz de darme una lista de Bases de datos más o menos conocidas, como los ficheros policiales para el DNI, que tienen los datos de todos los españoles, o la base de datos de alumnos de la Consejería de Educación, que gestiona con una aplicación de forma remota desde los centros, mientras que los datos se encuentran almacenados en un servidor de Sevilla. Elijo justamente estos dos **ejemplos** por una característica común: Se han diseñado para gestionar un gran volumen de información (más de 30 millones de registros en el caso del DNI, y no sé exactamente cuantos alumnos en toda Andalucía, pero desde luego muchísimos).

Las bases de datos no son un tipo de datos propiamente dicho, si no de una estructura que permite almacenar, consultar, actualizar y gestionar grandes cantidades de información, de forma cómoda y segura, y añadiendo funcionalidad extra al procesamiento que puede hacerse mediante ficheros individuales, ya **que mediante un sistema gestor de bases de datos podremos realizar cómodamente consultas en las que se vean involucrados campos de múltiples registros que están contenidos en distintos ficheros, por ejemplo**. De hecho, **la base de datos estará formada por múltiples ficheros de distintos tipos, pero relacionados entre sí.**

Aunque los lenguajes de programación proporcionan tipos de datos fichero, y toda una serie de funciones para permitir trabajar con ficheros individuales, la realidad es que la mayoría de las aplicaciones de gestión de hoy en día, sobre todo si se trata de aplicaciones de gestión, almacenan, consultan, mantienen, modifican y actualizan todos sus datos haciendo uso de **Bases de Datos**.

Además, los sistemas de gestión de bases de datos garantizan que el mantenimiento de esos datos es hasta cierto punto independiente de la aplicación que los usa. Es muy habitual que se cree una base de datos, y que sean varias las aplicaciones que acceden a los mismos datos, incluso aunque se trate de aplicaciones desarrolladas con distintos lenguajes de programación. Los datos de la Base de datos son independientes de la aplicación y del lenguaje usado, algo que no siempre ocurre cuando se trabaja con ficheros individuales.

Autoevaluación

Señala la afirmación correcta:

a) Una base de datos es un tipo de datos propiamente dicho.

- Permite almacenar , consultar, actualizar y gestionar grandes cantidades de información.
- b) La base de datos estará formada por múltiples ficheros de distintos tipos independientes entre sí .
- c) Las bases de datos rara vez se usan en aplicaciones de gestión, por lo que la mayoría de las aplicaciones usarán sólo ficheros.
- d) La mayoría de aplicaciones de gestión hoy en día usan bases de datos, y las empresas apenas si usan ficheros directamente para almacenar los datos de su negocio.

Comprobar

Datos: Tipos y Características