El trabajo con fechas siempre ha sido un punto de inflexión en la carrera de los programadores. Esto se debe en parte a que en versiones anteriores de **Java** era un poco confuso e inconsistente. Sin embargo, en la nueva versión se ha implementado una **API** que nos permite sacar el máximo provecho del lenguaje con métodos más claros y de más fácil manejo.  
  
Esto hace que trabajar con **formatos de fecha y hora** en *Java* no sea una tortura. Para ello debemos estudiar las nuevas formas y posibilidades que nos da el lenguaje, con el objetivo de convertir uno de los puntos que más confusiones genera, en algo que dominemos de forma adecuada.

Los nuevos métodos comunes de las clases Date/Time

Para simplificar el trabajo, el equipo de desarrollo de **Java** nos ha facilitado una cantidad de métodos comunes que podemos utilizar, tanto cuando trabajamos con *fechas*, como cuando trabajamos con *horas*. Ello hace que el esfuerzo del aprendizaje sea menor al tener un comportamiento similar para ambos tipos de objetos.  
  
Vamos a listar a continuación algunos de los métodos que más resaltan junto a una pequeña explicación de para que podemos utilizarlos, con ello podremos tener una idea general cuando pasemos a la parte práctica más adelante.

at

Este método nos permite hacer una combinación del objeto que lo invoca con otro objeto.

format

Nos permite utilizar algunos parámetros para producir como resultado una fecha u hora con un formato específico que necesitemos.

get

Nos permite obtener algún elemento que le especifiquemos.

is

Nos permite validar si un objeto es del tipo con el que hagamos la comparación.

plus

Nos da la posibilidad de hacer una suma al objeto que lo invoque y nos retorna una copia del mismo con el valor ya sumado.

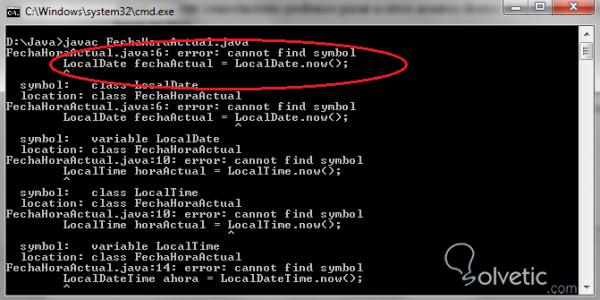
to

Nos permite hacer una conversión de un tipo a otro.

Ahora con este conocimiento podemos pasar a otros asuntos dentro de esta misma temática de fechas y horas en **Java**.

Calcular la fecha y hora actual

Como hemos venido explicando las formas que se utilizan en **Java** ahora han cambiado para la nueva versión, es por ello que para repetir los ejemplos que daremos necesitamos la [versión 8](http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html) del kit de desarrollo del lenguaje, de lo contrario recibiremos un error como el siguiente:

[](http://www.solvetic.com/uploads/monthly_11_2014/tutorials-2308-0-96393300-1417109390.jpg)

Aclarado el punto veamos a continuación que debemos hacer para obtener la información de la **fecha y hora actual** en Java.  
  
Para ello vamos a crear una clase en la cual vamos a instanciar objetos del tipo **LocalDate**, **LocalTime** y **LocalDateTime**. Cada uno de ellos nos dará respectivamente, la fecha actual, la hora actual y la fecha y hora actual.  
  
Esto es bastante útil en casos como los **logs de sistemas**, donde debemos capturar el momento preciso en el que se lleva una acción u ocurre algún evento que nos interese dejar registrado. Para lograr el objetivo podemos hacer un código similar al siguiente:

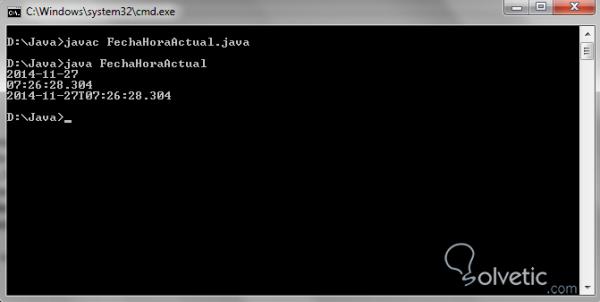
1. import java.time.\*;
2. public class FechaHoraActual {
3. public static void main(String[] args) {
4. LocalDate fechaActual = LocalDate.now();
5. System.out.println(fechaActual);
6. LocalTime horaActual = LocalTime.now();
7. System.out.println(horaActual);
8. LocalDateTime ahora = LocalDateTime.now();
9. System.out.println(ahora);
10. }
11. }

Como vemos lo único que hacemos es declarar nuestros objetos de los tipos ya antes mencionados y asignarles el valor que nos retorna con el método **now()**, el cuál es quien realmente se encarga de dar la fecha y horas actuales, luego solo nos preocupamos por imprimir cada una de los objetos declarados.

Importante

Es muy importante incluir el paquete **java.time** en su totalidad para que podamos acceder a todas las funciones que estamos utilizando, igualmente si tenemos dudas en la documentación oficial podemos encontrar mayor detalle de las opciones del mismo.

Veamos como luce la ejecución de nuestro programa:

[](http://www.solvetic.com/uploads/monthly_11_2014/tutorials-2308-0-52823400-1417109398.jpg)

Dar formato a las fechas

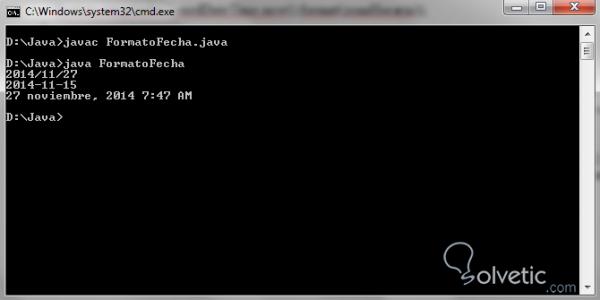
Otro aspecto que siempre debemos tomar en cuenta es la forma de mostrar la información. En materia de fechas tenemos muchas formas, sin embargo algunas son más convenientes que otras en algunas situaciones. Esto la mayoría de las veces nos lleva a realizar operaciones adicionales que nos hacen escribir líneas de código adicionales, lo cual en **Java 8** ya no es tan complejo.  
  
Según la aplicación que estemos desarrollando vamos a querer mostrar el mes por su número en el calendario o por su nombre, pero el nombre puede ser completo o abreviado igualmente el día o el año, todo ello nos da una cantidad muy amplia de opciones, todas válidas por igual. Para resolver este problema podremos utilizar el formateador de fechas.  
  
Para dar formato a las fechas tenemos algunos comodines o parámetros que nos permiten obtener los valores explicados en el párrafo anterior. Veamos cuales son los más importantes.

* **y**, nos permite acceder al año en formato de cuatro o dos dígitos (2014 o 14).
* **D**, nos permite obtener el número de día del año (225).
* **d,** al contrario del anterior nos devuelve el número del día del mes en cuestión (27).
* **L**, nos ayuda a obtener el mes del año en forma numérica, **M**nos da el mes en texto.
* **H**, nos da la hora.
* **s**, nos da los segundos.
* **m**, nos permite obtener los minutos.
* **a**, nos da el am o pm de la hora.
* **z,** nos permite acceder al nombre de la zona horaria.

Ya que conocemos los parámetros básicos de formato, vamos a intentar un nuevo ejercicio, en el siguiente código vamos a hacer algunos formatos, en primer lugar vamos a obtener la fecha actual y la vamos a convertir al formato **año/mes/día** con slashes en vez de guiones, luego vamos a hacer un parseo de una fecha estática que viene de un string con el mismo formato y finalmente vamos a mostrar una fecha completa incluyendo si es am o pm, para hacer uso de los comodines que hemos explicado anteriormente. Veamos el código correspondiente:

1. import java.time.\*;
2. import java.time.format.\*;
3. public class FormatoFecha {
4. public static void main(String[] args) {
5. DateTimeFormatter fechaFormateada = DateTimeFormatter.ofPattern("yyyy/LL/dd");
6. System.out.println(fechaFormateada.format(LocalDate.now()));
7. System.out.println(LocalDate.parse("2014/11/15", fechaFormateada));
8. DateTimeFormatter zonaHoraria = DateTimeFormatter.ofPattern("d MMMM, yyyy h:mm a");
9. System.out.println(ZonedDateTime.now().format(zonaHoraria));
10. }
11. }

Lo primero que podemos resaltar es que debemos importar tanto el paquete **java.time** con todas sus clases como el paquete **java.time.format**con todas sus clases también. Es posible hacer el ejercicio importando las clases específicas, pero eso lo dejamos de tarea, además así tienen una tarea para ver la documentación oficial.  
  
Luego vemos que creamos un objeto del tipo **DateTimeFormatter** y utilizamos el método **ofPattern()** que nos acepta los comodines especificados, vemos que colocamos los comodines por duplicado como **yyyy** para obtener las cuatro cifras del año en curso o también mm para obtener las dos cifras de los minutos.  
  
Finalmente imprimimos cada una de las fechas con sus formatos y obtenemos lo siguiente cuando ejecutamos nuestro programa:

[](http://www.solvetic.com/uploads/monthly_11_2014/tutorials-2308-0-10798900-1417109438.jpg)

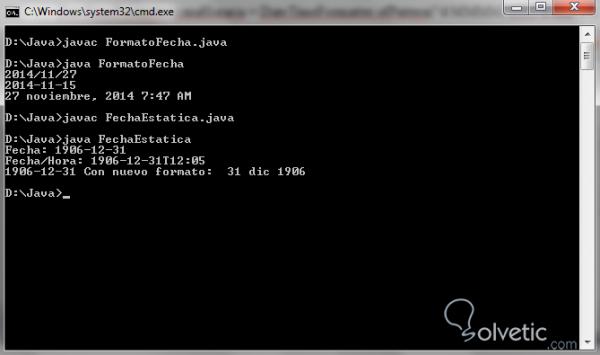
Aquí es donde podemos observar todos los cambios que hemos hecho a la forma de dar formato a las fechas, con ello podremos darle un mayor sentido y legibilidad tanto a nivel humano como para motores de *Base de Datos* como **MySQL** que necesita recibir la fecha por defecto en formato **yyyy-LL-dd**.

Utilizar fechas estáticas desde cadenas de texto

Otro uso que podemos darle a las nuevas clases de fecha y hora de **Java** es para poder utilizar fechas que vienen desde una cadena de texto, estás pueden ser creadas por un usuario, venir de un archivo de texto, etc. Pero lo importante es que debemos manipularlas y para ello podemos utilizar todas las herramientas que tenemos a mano.  
  
Veamos en el siguiente código como haciendo uso de lo que hemos visto podemos hacer cambios en una fecha proveniente de una cadena de texto.

1. import java.time.\*;
2. import java.time.format.\*;
3. public class FechaEstatica {
4. public static void main(String[] args) {
5. String fechaInicial = "1906-12-31";
6. LocalDate fechaTomada = LocalDate.parse(fechaInicial);
7. System.out.println("Fecha: " + fechaTomada);
8. String fechaInicialHora = "1906-12-31T12:05";
9. LocalDateTime fechaHoraTomada = LocalDateTime.parse(fechaInicialHora);
10. System.out.println("Fecha/Hora: " + fechaHoraTomada);
11. DateTimeFormatter df = DateTimeFormatter.ofPattern("dd MMM uuuu");
12. System.out.println(fechaTomada + " Con nuevo formato: " + df.format(fechaTomada));
13. }
14. }

En el código vemos como creamos una fecha inicial en cadena de texto, luego con un objeto del tipo **LocalDate** utilizamos el método **Parse** para incorporar la cadena de texto a un objeto tipo fecha, luego imprimimos el objeto y vemos que si tomó la fecha adecuadamente, repetimos el mismo procedimiento pero utilizando fecha y hora con **LocalDateTime**.  
  
Por último tomamos nuestro primer objeto creado y le aplicamos un nuevo formato, con ello confirmamos que hemos podido ingresar y manipular una fecha que establecimos desde una cadena de texto. Veamos como luce al ejecutar nuestro programa:

[](http://www.solvetic.com/uploads/monthly_11_2014/tutorials-2308-0-12266300-1417109614.jpg)

Con esto finalizamos este tutorial, hemos aprendido a trabajar con fechas en *Java* utilizando las mejoras en las clases que incorpora la nueva versión que es **Java 8**, esto nos puede servir para muchas aplicaciones a nivel de sistemas e incluso de interfaces de usuario.