

viewnext.adrformacion.com © ADR Infor SL
VICTOR TENA PALOMARES

Gestión del tiempo

© ADR Infor SL

viewnext.adrformacion.com © ADR Infor SL
VICTOR TENA PALOMARES

viewnext.adrformacion.com © ADR Infor SL
VICTOR TENA PALOMARES

Indice

Competencias y Resultados de Aprendizaje desarrollados en esta unidad	3
Gestión del tiempo	4
Objetivos	4
Gestión del tiempo en Java	4
Clases más utilizadas	4
Clase Date	4
Formato de fechas	9
Clase DateFormat	9
MÉTODOS	9
Clase Locale	12
MÉTODOS	13
Clase SimpleDateFormat	16
CONSTRUCTORES	16
MÉTODOS	18
Clase NumberFormat	20
MÉTODOS	20
Clase GregorianCalendar	23
CONSTRUCTORES	23
MÉTODOS	24
Ejercicios	30
Ejercicio 1. DateFormat	30
Recomendaciones	30
Datos a mostrar por consola	30
Ejercicio 2. Formatos numéricos	31
Datos a mostrar por consola	31
Ejercicio 3. GregorianCalendar	31
Datos a mostrar por consola	31
Ejercicio 4. Cálculos con fechas.	32
Lo que se necesita para comenzar	32
Recomendaciones	32
Datos a mostrar por consola	33
Recursos	34
Enlaces de Interés	34
Glosario.	34

Competencias y Resultados de Aprendizaje desarrollados en esta unidad

Competencia:

Gestionar el tiempo en programas Java

Resultados de Aprendizaje:

- Entender como se encarga Java de la gestión del tiempo.
- Entender la clase Date del paquete java.util
- Entender la clase Calendar del paquete java.util
- Entender la clase GregorianCalendar del paquete java.util
- Entender la clase TimeZone del paquete java.util
- Manejar la clase Date del paquete java.util
- Manejar la clase Calendar del paquete java.util
- Manejar la clase GregorianCalendar del paquete java.util

Gestión del tiempo

Objetivos

- Entender como se encarga Java de la gestión del tiempo.
- Entender la clase Date del paquete java.util
- Entender la clase Calendar del paquete java.util
- Entender la clase GregorianCalendar del paquete java.util
- Entender la clase TimeZone del paquete java.util

Gestión del tiempo en Java

Una necesidad primordial en el desarrollo de aplicaciones en la gestión de tiempo.

Para obtener y trabajar con información temporal en Java se emplean **cuatro clases del paquete java.util**:

- Date,
- Calendar,
- GregorianCalendar
- TimeZone.

Además de estas clases se emplean otras para aplicar formatos a la información temporal encapsulada en los objetos de las clases anteriores.

La mayoría de las **clases relacionadas con formatos pertenecen a java.text**. Las clases de este paquete que se van a estudiar en el tema son **DateFormat, SimpleDateFormat, DateFormatSymbols, NumberFormat y DecimalFormat**.

Clases más utilizadas

Clase Date

java.util.Date es una clase que se emplea para representar un instante de tiempo. Permite **capturar el instante de tiempo actual** en base a la configuración temporal de la máquina en la que se encuentra instalada la JVM (Java Virtual Machine).

Esta clase contiene muchos **métodos deprecados**, es decir, métodos **usados en versiones anteriores del J2SE** y que han sido sustituidos por otros, se supone mejores, en versiones más modernas.

Cuando se compila un código con métodos deprecados se **lanzan warnings**. Son avisos que **no impiden la ejecución** del código y que indican utilización en el código de métodos deprecados.

Ir a la API y observar la cantidad de métodos deprecados que contiene la clase Date.

CONSTRUCTORES

- **Date():** crea un objeto Date que contiene información sobre el instante en que se ejecuta la línea de código que contiene al constructor. La información horaria asociada a ese instante de tiempo depende de la configuración horaria de la máquina donde esté instalado el J2SE.
- **Date(long cantidadMilisegundos):** crea un objeto Date pasándole al argumento los milisegundos transcurridos desde el 1 de Enero de 1970 a las 00:00:00 GMT. Este instante de tiempo se denomina **epoch** y es el **origen de tiempos** que se emplea **en Java**. GMT son las siglas de Greenwich Mean Time o tiempo basado en el meridiano de Greenwich. En la península la hora es la de Greenwich más 1, es decir, GMT+01. Este método se utiliza para crear objetos Date referidos al pasado o al futuro, para conocer el tiempo transcurrido entre dos objetos Date, etc.

MÉTODOS

- **long getTime():** devuelve un entero con los milisegundos transcurridos desde el epoch hasta el momento en que se ejecuta la línea del constructor que crea el objeto Date sobre el que se aplica.

Ejemplo: muestra los milisegundos transcurridos desde el epoch hasta la fecha actual

Muestra los milisegundos transcurridos desde el epoch hasta la fecha actual

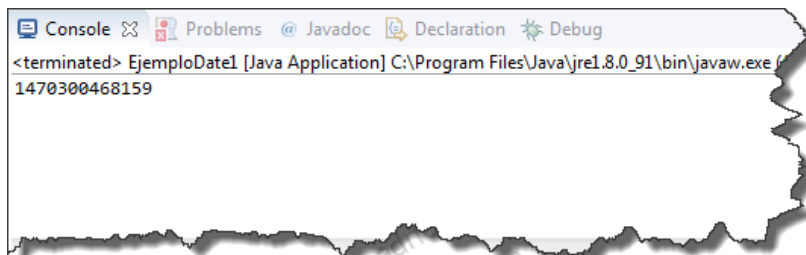
```

package unidad10.ejemplos;

import java.util.*;

public class EjemploDate1 {
    public static void main(String args[]) throws Exception{
        //Crear objeto Date con el primer constructor
        Date ahora=new Date();
        System.out.println(ahora.getTime());
    }
}

```

Por consola:

- **void setTime(long msg):** configura un objeto Date que apunta a la fecha asociada a los milisegundos que se le pasan al argumento. Estos milisegundos se miden respecto el epoch.

```

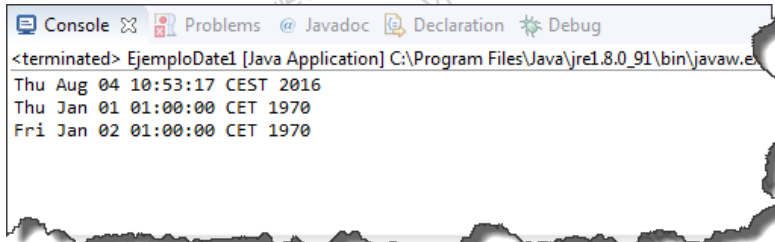
package unidad10.ejemplos;

import java.util.*;

public class EjemploDate1 {
    public static void main(String args[]) throws Exception{
        //Crear objeto Date con el primer constructor
        Date ahora=new Date();
        System.out.println(ahora);
        // objeto Date que apunta al epoch
        ahora.setTime(0);
        System.out.println(ahora);

        //Milisegundos contenidos en un día
        long msgUnDia = 24*60*60*1000;
        ahora.setTime(msgUnDia);
        System.out.println(ahora);
    }
}

```

Por consola:

```
<terminated> EjemploDate1 [Java Application] C:\Program Files\Java\jre1.8.0_91\bin\javaw.exe
Thu Aug 04 10:53:17 CEST 2016
Thu Jan 01 01:00:00 CET 1970
Fri Jan 02 01:00:00 CET 1970
```

- **boolean before(Date unaFecha):** devuelve true si el objeto Date sobre el que se aplica el método representa un instante de tiempo anterior al objeto Date que se le pasa al argumento. False en caso contrario.
- **boolean after(Date unaFecha):** devuelve true si el objeto Date sobre el que se aplica el método representa un instante de tiempo posterior al objeto Date que se le pasa al argumento. False en caso contrario.
- **boolean equals(Date unaFecha):** devuelve true si el objeto Date sobre el que se aplica el método equivale al Date que se le pasa al argumento. False en caso contrario.

Dos objetos Date son equivalentes cuando al aplicarles el método `getTime()`, devuelven el mismo número de milisegundos tomando como origen el epoch.

Se muestra un código que trabaja con aspectos básicos de la clase Date. Todos los códigos de este tema se guardarán en c:\cursojava\tema10 o en el workspace correspondiente, si se emplea IDE.

```
package unidad10.ejemplos;

import java.util.*;

public class EjemploDate1 {
    public static void main(String args[]) throws Exception{
        //Crear objeto Date con el primer constructor
        Date ahora=new Date();
        System.out.println("Fecha de ejecucion: "+ahora);

        //Obtener msg transcurridos desde el epoch hasta el instante de
        //ejecución de la línea de creación del objeto Date
        long msg=ahora.getTime();
        System.out.println("Milisegundos desde el epoch hasta la ejecucion"+
            " de este codigo= "+msg);

        //Crear otro objeto Date que almacena la misma información temporal
        //con el segundo constructor
        Date ahoraBis=new Date(msg);
        System.out.println("Fecha de ejecucion: "+ahoraBis);

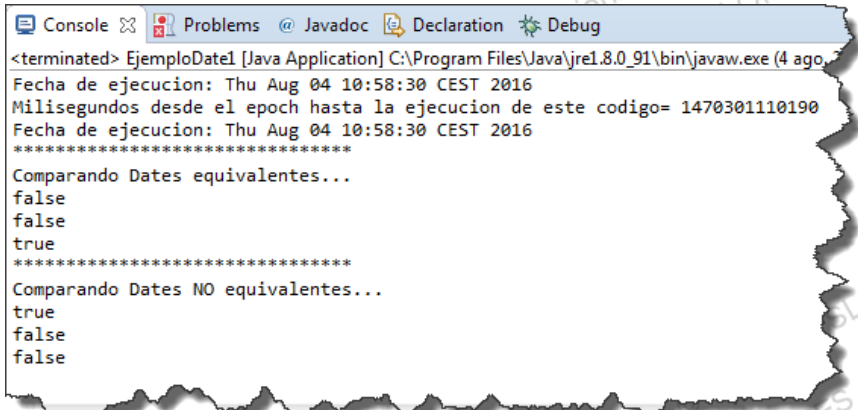
        //Comprobación de que los msg desde el epoch hasta el instante de
        //tiempo que representan los objetos Date ahora y ahoraBis son los
        //mismos
        System.out.println("*****");
        System.out.println("Comparando Dates equivalentes...");
        EjemploDate1.compararDates(ahora,ahoraBis);
        System.out.println("*****");

        //Se introduce un retardo de 1 segundo en la ejecución del código
        Thread.sleep(1000);

        //Crear objeto Date tras el retardo
        Date trasUnSegundo=new Date();
        System.out.println("Comparando Dates NO equivalentes...");
        EjemploDate1.compararDates(ahora,trasUnSegundo);
    }
    private static void compararDates(Date d1,Date d2){
        System.out.println(d1.before(d2));
        System.out.println(d1.after(d2));
        System.out.println(d1.equals(d2));
    }
}
```

Por consola:

com © ADR Infor SL
MARES



```

<terminated> EjemploDate1 [Java Application] C:\Program Files\Java\jre1.8.0_91\bin\javaw.exe (4 ago. 2016)
Fecha de ejecucion: Thu Aug 04 10:58:30 CEST 2016
Milisegundos desde el epoch hasta la ejecucion de este codigo= 1470301110190
Fecha de ejecucion: Thu Aug 04 10:58:30 CEST 2016
*****
Comparando Dates equivalentes...
false
false
true
*****
Comparando Dates NO equivalentes...
true
false
false

```

Formato de fechas

Para dar formato a las fechas con las que se va a trabajar se utilizan, entre otras, las siguientes clases:

- `java.text.DateFormat`
- `java.text.SimpleDateFormat` (subclase de `SimpleFormat`)
- `java.util.Locale`

Clase DateFormat

`java.text.DateFormat` es una clase **abstracta** que **modela formatos de fechas**. Al ser abstracta **no puede instanciarse**. Su misión, habitualmente, es definir métodos que usarán sus subclases en función del comportamiento que se desee para las mismas.

Existe un método estático "**DateFormat getInstance()**" que devuelve un objeto `DateFormat`.

Este tipo de método suele ser muy habitual en las clases abstractas. En la clase `Calendar` también aparece.

MÉTODOS

En este apartado se van a explicar algunos de los métodos más utilizados. **Para ver todos ir a la API.**

static DateFormat getInstance():

devuelve un DateFormat, que encapsula **información horaria y de fecha** (día, mes y año) en el formato del país que tenga establecida la máquina donde está instalada la JVM y con un estilo por defecto de tipo SHORT. SHORT es una variable de campo estática de DateFormat. Ir a la API y observar que esta clase cuenta con muchas variables de este estilo.

static DateFormat getTimeInstance(int estiloHorario, int estiloFecha, Locale país):

devuelve un DateFormat, que contiene **información horaria y de tipo fecha** (día, mes y año), con los estilos de hora y de fecha especificados en los dos primeros argumentos y en el formato del país o comunidad política indicada en el tercer argumento (la clase Locale se va a comentar en el siguiente apartado). El estilo suele configurarse con variables de campo estáticas como SHORT, MEDIUM, LONG, etc.

static DateFormat getDateInstance(int estiloHorario, Locale país):

ídem anterior, pero asociado a la **información horaria**.

static DateFormat getDateInstance(int estiloFecha, Locale país):

ídem anterior, pero asociado a la **información de tipo fecha** (del día, mes y año).

String format(Date fechaGlobal):

devuelve una String que encapsula la información temporal con el formato deseado. Es el método que debe aplicarse sobre el DateFormat que almacena el formato para obtener la información temporal formateada. **MUY USADO.**

```

package unidad10.ejemplos;

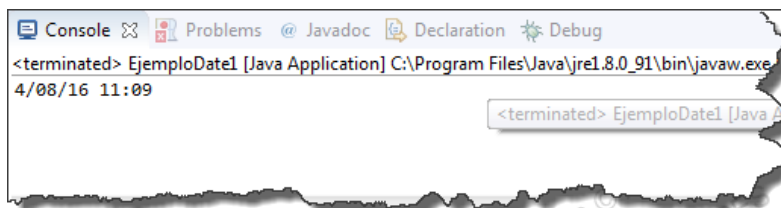
import java.text.DateFormat;
import java.util.*;

public class EjemploDate1 {
    public static void main(String args[]) throws Exception{
        //Crear objeto Date con el primer constructor
        Date ahora=new Date();
        DateFormat df = DateFormat.getInstance();

        //Fecha formateada en castellano estilo short.
        String fechaFormateada = df.format(ahora);
        System.out.println(fechaFormateada);
    }
}

```

Por consola:



Si el ordenador en el que se ejecuta este código tuviera como configuración de país,

Alemania 4.08.16 11:09

Estados Unidos 4/8/16 11:54 PM

Date parse(String fechaTextual):

devuelve un objeto Date generado en base a la cadena de texto que se le pasa al argumento. Si esa cadena no concuerda con el formato del DateFormat sobre el que se aplica, se lanza una ParseException. Esta excepción es de gestión obligatoria, así que hay que considerarla en un bloque try ... catch o con la cláusula throws.

```

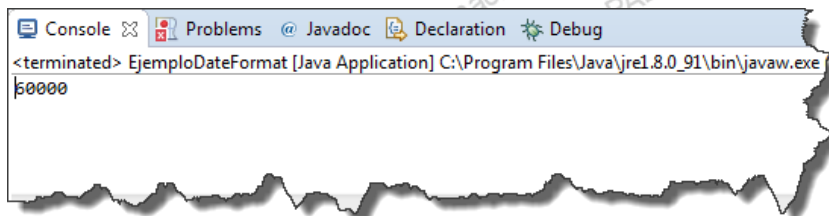
package unidad10.ejemplos;

import java.text.DateFormat;
import java.text.ParseException;
import java.util.Date;

public class EjemploDateFormat{
    public static void main(String args[]) throws Exception{
        DateFormat df = DateFormat.getInstance();
        /*
         * Si las cadenas de texto no se adaptan al formato
         * del DateFormat se lanza una ParseException.
         */
        String fecha1 = "22/03/80 13:55";
        String fecha2 = "22/03/80 13:54";
        try{
            Date d1 = df.parse(fecha1);
            Date d2 = df.parse(fecha2);
            System.out.println(d1.getTime()-d2.getTime());
        } catch (ParseException e) {
            System.out.println("Dato de fecha incorrecto");
        }
    }
}

```

Por consola:



El valor se muestra siempre en milisegundos a no ser que el desarrollador lo modifique.

Clase Locale

java.util.Locale es una clase que **modela** a un país o, hablando más ampliamente, a **una región con una cultura** más o menos **común**. Posee muchas variables de campo estáticas (ir a la API para echarles un vistazo). Desafortunadamente no hay ninguna que represente a nuestro país (paciencia :-()).

MÉTODOS

Se explican algunos. Para ver todos ir a la API.

static Locale getDefault():

devuelve el Locale asociado al “país” de la máquina donde se ejecuta el código y se encuentra instalada la JVM. Depende de la configuración de sistema operativo establecida.

String getLanguage():

devuelve el código de la lengua del país asociado al Locale sobre el que se aplica.

String getCountry():

ídem, pero del país.

String getDisplayLanguage():

devuelve el nombre de la lengua asociada al Locale sobre el que se aplica, expresada en la lengua del Locale por defecto.

String getDisplayCountry():

ídem anterior, pero del país.

String getISO3Language():

devuelve el código ISO de la lengua del país asociado al Locale sobre el que se aplica.

String getISO3Country():

ídem anterior, pero del país.

Se muestra un código que trabaja con aspectos básicos de la clase Locale.

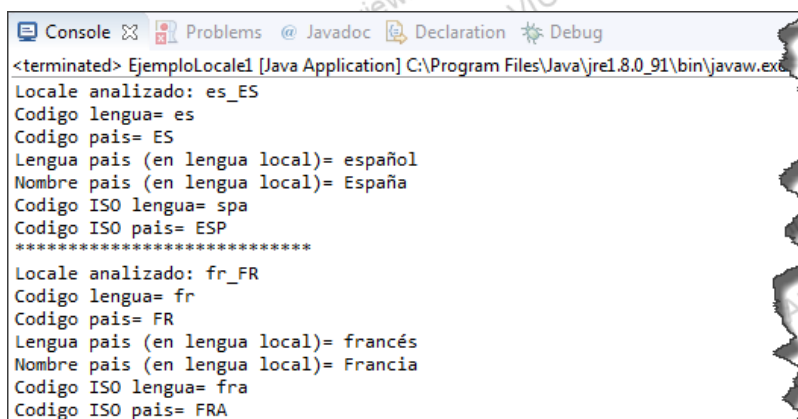
```
package unidad10.ejemplos;
import java.util.*;

public class EjemploLocale1 {
    public static void main(String args[]){
        EjemploLocale1 e11=new EjemploLocale1();

        //Locale asociado a la máquina donde se ejecuta
        Locale loc=Locale.getDefault();
        e11.informacion(loc);
        System.out.println("*****");

        //Locale asociado a Francia. Se usa variable de campo estática
        loc=Locale.FRANCE;
        e11.informacion(loc);
    }
    public void informacion(Locale loc){
        System.out.println("Locale analizado: "+loc);
        System.out.println("Codigo lengua= "+loc.getLanguage());
        System.out.println("Codigo pais= "+loc.getCountry());
        System.out.println("Lengua pais (en lengua local)= "+
            loc.getDisplayLanguage());
        System.out.println("Nombre pais (en lengua local)= "+
            loc.getDisplayCountry());
        System.out.println("Codigo ISO lengua= "+loc.getISO3Language());
        System.out.println("Codigo ISO pais= "+loc.getISO3Country());
    }
}
```

Por consola:



```
<terminated> EjemploLocale1 [Java Application] C:\Program Files\Java\jre1.8.0_91\bin\javaw.exe
Locale analizado: es_ES
Codigo lengua= es
Codigo pais= ES
Lengua pais (en lengua local)= español
Nombre pais (en lengua local)= España
Codigo ISO lengua= spa
Codigo ISO pais= ESP
*****
Locale analizado: fr_FR
Codigo lengua= fr
Codigo pais= FR
Lengua pais (en lengua local)= francés
Nombre pais (en lengua local)= Francia
Codigo ISO lengua= fra
Codigo ISO pais= FRA
```

Se muestra un código que trabaja con aspectos básicos de las clases **Locale**, **DateFormat** y algunas de sus variables de campo estáticas como **SHORT**, **MEDIUM** y **LONG**.

```
package unidad10.ejemplos;
import java.text.*;
import java.util.*;

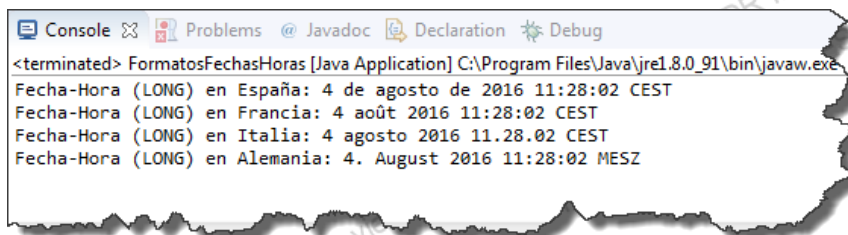
public class FormatosFechasHoras {
    public static void main(String args[]){
        FormatosFechasHoras ffh=new FormatosFechasHoras();

        //Crear objeto Date
        Date ahora=new Date();

        //Construcción de cuatro DateFormat asociados a España, Francia,
        //Italia y Alemania
        DateFormat dfEspañol=DateFormat.getDateInstance(DateFormat.LONG,
            DateFormat.LONG,Locale.getDefault());
        DateFormat dfFrances=DateFormat.getDateInstance(DateFormat.LONG,
            DateFormat.LONG,Locale.FRANCE);
        DateFormat dfItaliano=DateFormat.getDateInstance(DateFormat.LONG,
            DateFormat.LONG,Locale.ITALY);
        DateFormat dfAleman=DateFormat.getDateInstance(DateFormat.LONG,
            DateFormat.LONG,Locale.GERMANY);

        //Almacenar los DateFormat en un array
        DateFormat formatos[]={dfEspañol, dfFrances, dfItaliano, dfAleman};
        ffh.mostrarFechaHora(formatos,ahora);
    }
    public void mostrarFechaHora(DateFormat formatos[],Date ahora){
        Locale loc[]={Locale.getDefault(),Locale. FRANCE,Locale.ITALY,
            Locale.GERMANY};
        for(int i=0;i<formatos.length;i++)
            System.out.println("Fecha-Hora (LONG) en "+
                loc[i].getDisplayCountry()+"": "+formatos[i].format(ahora));
    }
}
```

Por consola:



```
<terminated> FormatosFechasHoras [Java Application] C:\Program Files\Java\jre1.8.0_91\bin\javaw.exe
Fecha-Hora (LONG) en España: 4 de agosto de 2016 11:28:02 CEST
Fecha-Hora (LONG) en Francia: 4 août 2016 11:28:02 CEST
Fecha-Hora (LONG) en Italia: 4 agosto 2016 11.28.02 CEST
Fecha-Hora (LONG) en Alemania: 4. August 2016 11:28:02 MESZ
```

Si se emplea la variable de campo estática MEDIUM en vez de LONG, al construir los DateFormat:

Fecha-Hora (MEDIUM) en España: 05-ene-03 13:42:15 CET

Fecha-Hora (MEDIUM) en Francia: 5 janv. 03 13:42:15 CET

Fecha-Hora (MEDIUM) en Italia: 5-gen-03 13.42.15 CET

Fecha-Hora (MEDIUM) en Alemania: 05.01.2003 13:42:15 CET

Si se emplea la variable de campo estática SHORT:

Fecha-Hora (SHORT) en España: 5/01/03 13:43:26 CET

Fecha-Hora (SHORT) en Francia: 05/01/03 13:43:26 CET

Fecha-Hora (SHORT) en Italia: 05/01/03 13.43.26 CET

Fecha-Hora (SHORT) en Alemania: 05.01.03 13:43:26 CET

Antes de continuar, se recomienda hacer el primer ejercicio del tema.

Clase SimpleDateFormat

`java.text.SimpleDateFormat` es una **subclase de `DateFormat`** que permite **definir formatos o patrones de fecha y hora más especializados** que los de su superclase. Por ejemplo, se puede **mostrar el día de la semana, los nombres completos de los meses, etc.**

CONSTRUCTORES

`SimpleDateFormat():`

crea un objeto `SimpleDateFormat` en base al formato o patrón por defecto asociado al `Locale` por defecto.

`SimpleDateFormat(String formato):`

ídem anterior, pero en base al formato del argumento.

En los formatos debe tenerse en cuenta que:
d equivale a el día del mes

M al mes (el formato del ejemplo tiene 10 “emes” porque el máximo número de letras de un mes en castellano puede ser 10 (septiembre))

y al año

E al día de la semana (el formato del ejemplo tiene 9 “es” porque el máximo número de letras de un día de la semana en castellano es 9 (miércoles))

h a las horas en formato simple

H a las horas en formato 24 horas

m a los minutos

s a los segundos

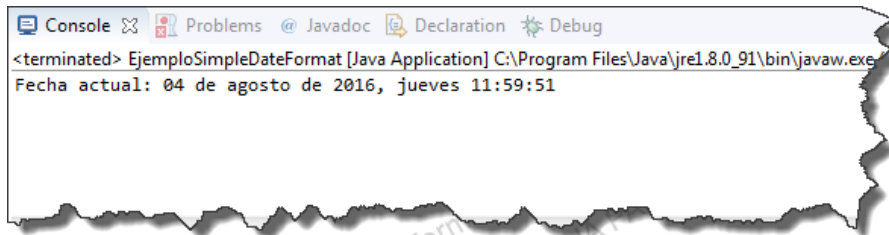
Ahora vamos a ver un par de ejemplos:

```
package unidad10.ejemplos;
import java.text.*;
import java.util.*;

public class EjemploSimpleDateFormat {
    public static void main(String args[]){
        EjemploSimpleDateFormat esdf=new EjemploSimpleDateFormat();

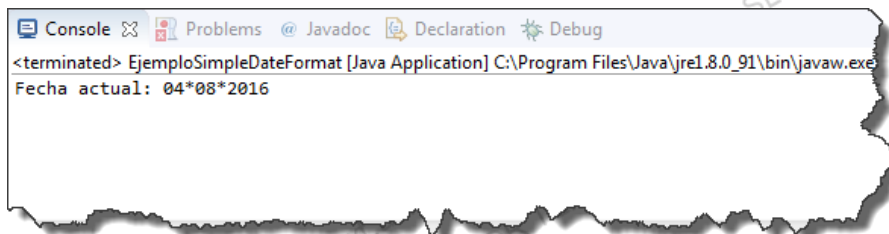
        Date ahora=new Date();

        /*
         * Los caracteres que no son de formato se escriben
         * entre comillas simples.
         */
        String formato="dd 'de' MMMMMMMMMMMM 'de' yyyy, EEEEEEEEE HH
:mm:ss";
        SimpleDateFormat sdf = new SimpleDateFormat(formato);
        System.out.println("Fecha actual: " + sdf.format(ahora));
    }
}
```

Por consola:**Ejemplo 2:**

```
package unidad10.ejemplos;
import java.text.*;
import java.util.*;

public class EjemploSimpleDateFormat{
    public static void main(String args[]){
        Date ahora=new Date();
        String formato="dd*MM*yyyy";
        SimpleDateFormat sdf= new SimpleDateFormat(formato);
        System.out.println("Fecha actual: " + sdf.format(ahora));
    }
}
```

Por consola:**SimpleDateFormat(String formato, Locale pais):**

Ídem anterior, pero especificando en el segundo argumento el Locale deseado.

MÉTODOS

Como en otros apartados del curso, no se van estudiar todos los métodos, veremos los más utilizados. En este caso solamente veremos un método.

Para ver todos ir a la API.

- **void applyPattern(String formato):** permite aplicar el formato o patrón pasado al argumento cuando se aplica sobre un SimpleDateFormat creado con el primer constructor.

Se escribe un código que muestra la fecha completa en varios idiomas con formatos o patrones personalizados.

```
package unidad10.ejemplos;
import java.text.*;
import java.util.*;

public class EjemploSimpleDateFormat1 {
    public static void main(String args[]){
        EjemploSimpleDateFormat1 esdf=new EjemploSimpleDateFormat1();

        Date ahora=new Date();

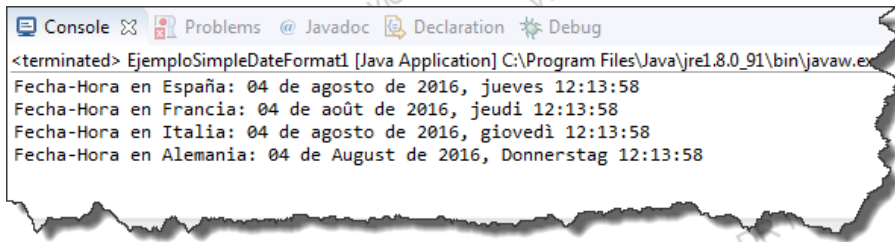
        //Definir formato
        String formato="dd 'de' MMMMMMMMMM 'de' yyyy, EEEEEEEEE HH:mm:ss";

        //Construcción de cuatro SimpleDateFormat asociados a España,
        //Francia, Italia y Alemania
        SimpleDateFormat sdfEspañol=new SimpleDateFormat(formato,
            Locale.getDefault());
        SimpleDateFormat sdfFrances=new SimpleDateFormat(formato,
            Locale.FRANCE);
        SimpleDateFormat sdfItaliano=new SimpleDateFormat(formato,
            Locale.ITALY);
        SimpleDateFormat sdfAleman=new SimpleDateFormat(formato,
            Locale.GERMANY);

        //Almacenar los SimpleDateFormat en un array
        SimpleDateFormat formatos[]={sdfEspañol,sdfFrances,sdfItaliano,
            sdfAleman};
        esdf.mostrarFechaHora(formatos,ahora);
    }
    public void mostrarFechaHora(SimpleDateFormat formatos[],Date ahora){
        Locale loc[]={Locale.getDefault(),Locale.FRANCE,Locale.ITALY,
            Locale.GERMANY};
        for(int i=0;i<formatos.length;i++)
            System.out.println("Fecha-Hora en "+loc[i].getDisplayCountry()+
                ": "+formatos[i].format(ahora));
    }
}
```

Por consola:

viewnext.adrformacion.com © ADR Infor SL
VICTOR TENA PALOMARES



```

<terminated> EjemploSimpleDateFormat1 [Java Application] C:\Program Files\Java\jre1.8.0_91\bin\javaw.exe
Fecha-Hora en España: 04 de agosto de 2016, jueves 12:13:58
Fecha-Hora en Francia: 04 de août de 2016, jeudi 12:13:58
Fecha-Hora en Italia: 04 de agosto de 2016, giovedì 12:13:58
Fecha-Hora en Alemania: 04 de August de 2016, Donnerstag 12:13:58

```

Clase NumberFormat

java.text.NumberFormat es una clase **abstracta** muy similar a **DateFormat** que **modela formatos de números**. Por defecto, si los números son decimales y se formatean, se redondean en base a la tercera cifra decimal.

Considerando que es 3 el número máximo de cifras decimales, si el cuarto decimal es mayor o igual que 5 se aumenta en una unidad la tercera cifra decimal.

36.1235 se aproxima a 36.124

Si, en cambio, es menor que 5 permanece igual la tercera cifra decimal.

36.1234 se aproxima a 36.123

MÉTODOS

Se explican los más utilizados.

Para ver todos ir a la API.

static NumberFormat getInstance(Locale país):

crea una **NumberFormat** con el formato de número general asociado al **Locale** del argumento. Existe una versión sobrecargada de este método sin ningún argumento que indica uso del **Locale** local. Ídem para los dos métodos siguientes.

static NumberFormat getCurrencyInstance(Locale país):

crea un **NumberFormat** con el formato de moneda asociado al **Locale** del argumento.

static NumberFormat getPercentInstance(Locale país):

crea un NumberFormat con el formato de porcentaje asociado al Locale del argumento.

int getMaximumFractionDigits():

devuelve el número máximo de cifras decimales con el que se va a representar un número decimal. Su valor como se comentó al principio de la sección es 3.

void setMaximumFractionDigits(int maxDecimales):

configura el número máximo de cifras decimales con el que se va a representar a un número decimal.

String format(double num):

devuelve una String que muestra el número decimal que se le pasa al argumento con la información de formato del NumberFormat sobre el que se aplica. Es el mismo que el que se emplea en DecimalFormat.

String format(long num):

ídem anterior, pero trabajando con números enteros.

Muestra los formatos de número general, de moneda y de porcentaje asociados al Locale local y a otros de un número entero y de otro decimal

```
package unidad10.ejemplos;
import java.text.*;
import java.util.*;

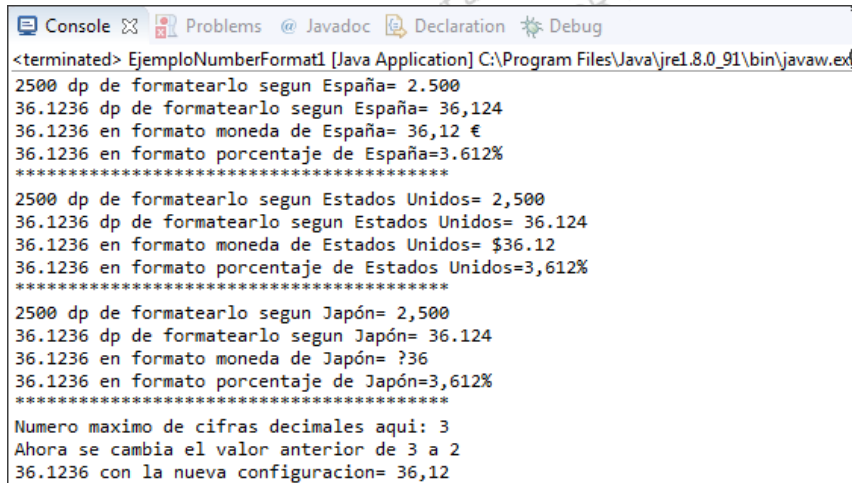
public class EjemploNumberFormat1 {
    public static void main(String args[]) {
        long entero=2500L;
        double decimal=36.1236;
        NumberFormat nfGeneral,nfMoneda,nfPorcentaje;
        Locale loc[]={Locale.getDefault(),Locale.US,Locale.JAPAN};
        for(int i=0;i<loc.length;i++){
            nfGeneral=NumberFormat.getNumberInstance(loc[i]);
            System.out.println(entero+" dp de formatearlo segun "+
                loc[i].getDisplayCountry()+"="+nfGeneral.format(entero));
            System.out.println(decimal+" dp de formatearlo segun "+
                loc[i].getDisplayCountry()+"="+nfGeneral.format(decimal));

            nfMoneda=NumberFormat.getCurrencyInstance(loc[i]);
            System.out.println(decimal+" en formato moneda de "+
                loc[i].getDisplayCountry()+"="+nfMoneda.format(decimal));

            nfPorcentaje=NumberFormat.getPercentInstance(loc[i]);
            System.out.println(decimal+" en formato porcentaje de "+
                loc[i].getDisplayCountry()+"="+nfPorcentaje.format(decimal));
            System.out.println("*****");
        }

        NumberFormat nfEspañol=NumberFormat.getNumberInstance(
            Locale.getDefault());
        System.out.println("Numero maximo de cifras decimales aqui: "+
            nfEspañol.getMaximumFractionDigits());
        System.out.println("Ahora se cambia el valor anterior de 3 a 2");
        nfEspañol.setMaximumFractionDigits(2);
        System.out.println(decimal+" con la nueva configuracion= "+
            nfEspañol.format(decimal));
    }
}
```

Por consola:



```

<terminated> EjemploNumberFormat1 [Java Application] C:\Program Files\Java\jre1.8.0_91\bin\javaw.exe
2500 dp de formatearlo segun España= 2.500
36.1236 dp de formatearlo segun España= 36,124
36.1236 en formato moneda de España= 36,12 €
36.1236 en formato porcentaje de España=3.612%
*****
2500 dp de formatearlo segun Estados Unidos= 2,500
36.1236 dp de formatearlo segun Estados Unidos= 36.124
36.1236 en formato moneda de Estados Unidos= $36.12
36.1236 en formato porcentaje de Estados Unidos=3,612%
*****
2500 dp de formatearlo segun Japón= 2,500
36.1236 dp de formatearlo segun Japón= 36.124
36.1236 en formato moneda de Japón= ¥36
36.1236 en formato porcentaje de Japón=3,612%
*****
Numero maximo de cifras decimales aqui: 3
Ahora se cambia el valor anterior de 3 a 2
36.1236 con la nueva configuracion= 36,12

```

Es una **subclase de NumberFormat** que permite crear formatos personalizados de números entre otras muchas cosas más. **Se deja al alumno su estudio.**

Antes de continuar, se recomienda hacer el segundo ejercicio del tema.

Clase GregorianCalendar

java.util.GregorianCalendar es una **subclase** de la clase abstracta **Calendar** que **modela calendarios** referidos al presente, pasado o futuro.

Con un objeto de esta clase es posible averiguar el día de la semana en que una persona nació, el día de la semana en que caerá la Navidad del año 2025, comprobar si un año ha sido, es o será bisiesto, contabilizar cuántos días del año actual han transcurrido, etc.

Esta clase **cuenta con muchas variables de campo estáticas heredadas de Calendar**, así que **conviene echar un vistazo a la API** para familiarizarse con ellas.

CONSTRUCTORES

GregorianCalendar():

crea un **GregorianCalendar** u objeto que simula a un calendario tomando como referencia el instante de tiempo actual, es decir, crea un calendario del año actual.

Aparte de este constructor, puede crearse un calendario del año actual mediante los métodos de Calendar:

- static Calendar getInstance()
- void setTime(Date fecha)

```
Calendar cal=Calendar.getInstance();
```

```
cal.setTime(new Date());
```

Las dos líneas anteriores son equivalentes a:

```
GregorianCalendar cal=new GregorianCalendar();
```

GregorianCalendar(int año, int mes, int día):

crea un calendario en base a los tres argumentos que se le pasan.

crear un calendario del año actual empleando las variables de campo estáticas de Calendar YEAR, MONTH y DAY_OF_MONTH. Luego otro en base al día de Navidad del 2005.

```
package unidad10.ejemplos;
```

```
import java.util.*;
```

```
public class EjemploGregorianCalendar {
    public static void main(String args[]) {
        // Crear un calendario actual
        GregorianCalendar gc = new GregorianCalendar(Calendar.YEAR, Calendar.MONTH, Calendar.DAY_OF_MONTH);
        GregorianCalendar gcFuturo = new GregorianCalendar(2015, Calendar.DECEMBER, 25);
    }
}
```

GregorianCalendar(int año, int mes, int día, int hora, int minuto, int segundo):

ídem anterior, pero con precisión total

MÉTODOS

Se explican algunos. Para ver todos ir a la API. Muchos de estos métodos se emplean con variables de campo estáticas de su superclase Calendar. Estas variables están asociadas a propiedades temporales muy características de un calendario y devuelven números enteros.

boolean isLeapYear(int año):

devuelve true si el año que se le pasa al argumento es bisiesto.

void set(int propTemp, int valor):

es un método de su superclase Calendar. Se emplea para configurar el año, mes, día, etc. de un calendario.

El primer argumento indica la propiedad temporal que va a configurarse. Suele ser una variable de campo estática

El segundo argumento se emplea para especificar su valor.

```
package unidad10.ejemplos;

import java.util.*;

public class EjemploGregorianCalendar {
    public static void main(String args[]) {
        // Crear un calendario actual
        GregorianCalendar gc = new GregorianCalendar();
        gc.set(Calendar.YEAR, 2005);
        gc.set(Calendar.MONTH, 5);
        gc.set(Calendar.DAY_OF_MONTH, 25);
    }
}
```

Es equivalente a `GregorianCalendar gc=new GregorianCalendar(2002,5,18);`

int get(int propTemp):

otro método de su superclase. Devuelve un entero que contiene información de la propiedad temporal asociada a la variable de campo estática que se le pasa al argumento. MUY USADO.

los números enteros asociados a los meses del año y a los días de la semana son:

MESES: Enero-->0, Febrero-->1, ..., Diciembre-->11

DÍAS SEMANA: Domingo-->1, Lunes-->2, ..., Sábado-->7

```
String[] getMonths()
```

```
String[] getWeekdays()
```

Date getTime():

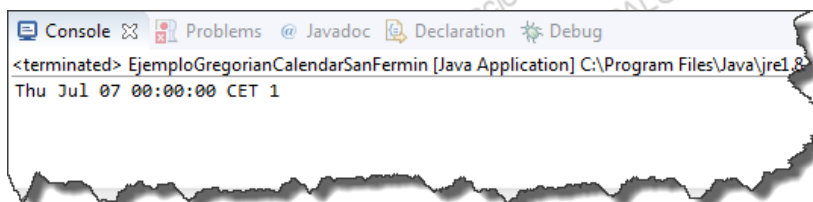
otro método de su superclase. Devuelve un objeto Date que se genera en base a la información temporal del calendario sobre el que se aplica. **MUY USADO.**

```
package unidad10.ejemplos;

import java.util.*;

public class EjemploGregorianCalendarSanFermin {
    public static void main(String args[]) {
        // Crear un calendario actual
        GregorianCalendar gcSanFermin = new GregorianCalendar(
            Calendar.YEAR, Calendar.JULY, 7);
        Date fechaSanFermin = gcSanFermin.getTime();
        System.out.println(fechaSanFermin);
    }
}
```

Por consola:



void setTime(Date fecha):

también es un método de Calendar. Configura el calendario sobre el que se aplica a la fecha que se le pasa al argumento. **MUY USADO.**

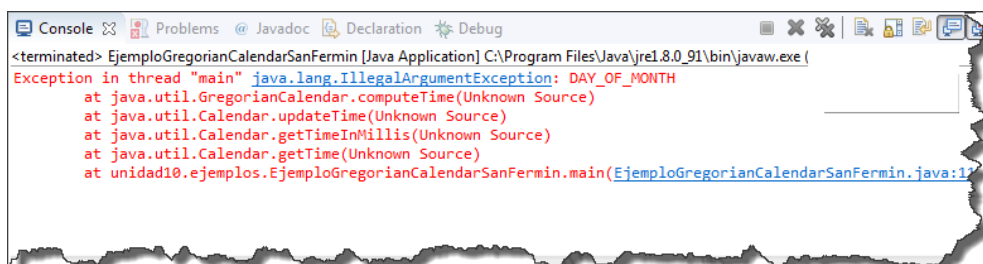
void setLenient(boolean b):

otro método de Calendar. Si se le pasa false configura el calendario de modo que no admita fechas incorrectas. Ejemplo: `GregorianCalendar(Calendar.YEAR, Calendar.JULY, 45)`. Compila bien, pero al ejecutar se lanza una `java.lang.IllegalArgumentException`. Lenient significa indulgente. Por defecto todo calendario es indulgente o permisivo con las fechas incorrectas.

```
package unidad10.ejemplos;

import java.util.*;

public class EjemploGregorianCalendarSanFermin {
    public static void main(String args[]) {
        // Crear un calendario actual
        GregorianCalendar gcSanFermin = new GregorianCalendar(
            Calendar.YEAR, Calendar.JULY, 45);
        gcSanFermin.setLenient(false);
        Date fechaSanFermin = gcSanFermin.getTime();
        System.out.println(fechaSanFermin);
    }
}
```

Por consola:


```
<terminated> EjemploGregorianCalendarSanFermin [Java Application] C:\Program Files\Java\jre1.8.0_91\bin\javaw.exe {
Exception in thread "main" java.lang.IllegalArgumentException: DAY_OF_MONTH
    at java.util.GregorianCalendar.computeTime(Unknown Source)
    at java.util.Calendar.updateTime(Unknown Source)
    at java.util.Calendar.getTimeInMillis(Unknown Source)
    at java.util.Calendar.getTime(Unknown Source)
    at unidad10.ejemplos.EjemploGregorianCalendarSanFermin.main(EjemploGregorianCalendarSanFermin.java:17)
```

Muestra cómo obtener la información que encapsulan algunas de las propiedades temporales asociadas a un `GregorianCalendar` actual.

```
package unidad10.ejemplos;
import java.util.*;

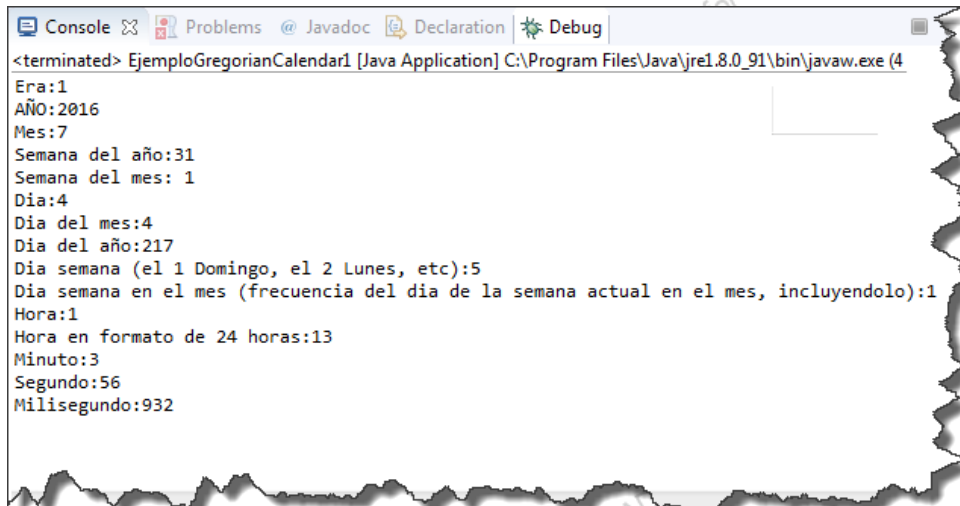
public class EjemploGregorianCalendar1 {
    public static void main(String args[]){
        //Crear un calendario actual
        GregorianCalendar gc=new GregorianCalendar();

        //Mostrar información
        System.out.println("Era:"+gc.get(Calendar.ERA));
        System.out.println("AÑO:"+gc.get(Calendar.YEAR));
        System.out.println("Mes:"+gc.get(Calendar.MONTH));
        System.out.println("Semana del año:"+
            gc.get(Calendar.WEEK_OF_YEAR));
        System.out.println("Semana del mes: "+
            gc.get(Calendar.WEEK_OF_MONTH));
        System.out.println("Dia:"+gc.get(Calendar.DATE));
        System.out.println("Dia del mes:"+gc.get(Calendar.DAY_OF_M
ONTH));
        System.out.println("Dia del año:"+
            gc.get(Calendar.DAY_OF_YEAR));
        System.out.println("Dia semana (el 1 Domingo, el 2 Lunes, etc):"+
            gc.get(Calendar.DAY_OF_WEEK));
        System.out.println("Dia semana en el mes (frecuencia del dia de "+
            "la semana actual en el mes, incluyendolo):"+
            gc.get(Calendar.DAY_OF_WEEK_IN_MONTH));
        System.out.println("Hora:"+gc.get(Calendar.HOUR));
        System.out.println("Hora en formato de 24 horas:"+
            gc.get(Calendar.HOUR_OF_DAY));
        System.out.println("Minuto:"+gc.get(Calendar.MINUTE));
        System.out.println("Segundo:"+gc.get(Calendar.SECOND));
        System.out.println("Milisegundo:"+gc.get(Calendar.MILLISECO
ND));
    }
}
```

Por consola:

viewnext.adrformacion.com © ADR
VICTOR TENA PALOMARES

SL



```
<terminated> EjemploGregorianCalendar1 [Java Application] C:\Program Files\Java\jre1.8.0_91\bin\javaw.exe (4)
Era:1
AÑO:2016
Mes:7
Semana del año:31
Semana del mes: 1
Día:4
Día del mes:4
Día del año:217
Día semana (el 1 Domingo, el 2 Lunes, etc):5
Día semana en el mes (frecuencia del día de la semana actual en el mes, incluyendolo):1
Hora:1
Hora en formato de 24 horas:13
Minuto:3
Segundo:56
Milisegundo:932
```

¿Cómo se modificaría el código anterior para que muestre el mes del año y el día de la semana con sus nombres?

Pista: usar la clase DateFormatSymbols

Ahora conviene realizar el resto de ejercicios del tema

Ejercicios

Ejercicio 1. DateFormat

15

Realizar un programa que contenga una clase pública de nombre **MesesDiasSemana**.

Solicitará al usuario un idioma (alemán, francés, italiano o castellano) y mostrará por consola los meses del año y los días de la semana (comenzando por el domingo) en ese idioma.

Recomendaciones

Usar los métodos de la clase `java.text.DateFormatSymbols` "`String[] getMonths()`" y "`String[] getWeekdays()`".

Consultar la API para echar un vistazo a sus constructores y a la información que proporciona un objeto `DateFormatSymbols`.

Estos arrays encapsulan información de los meses del año y los días de la semana asociados a un `Locale`. Su número de elementos es 13 y 8 respectivamente pues contemplan el uso de calendarios lunares.

En Occidente, un año tiene 12 meses y una semana 7 días, de modo que un elemento en los dos arrays queda vacío.

- En el de los meses es el último. A enero-->0, febrero-->1, ..., diciembre-->11
- En el de los días de la semana, es el primero. Al domingo-->1, lunes-->2, ..., sábado-->7

Suele tenerse en cuenta a la hora de recorrerlos con bucles:

- En el de los meses se va desde 0 hasta `length - 1`
- En el de los días de la semana, desde 1 hasta `length`

Datos a mostrar por consola

```
<terminated> MesesDiasSemana [Java Application] C:\Program Files\Java\jre1.8.0_91\bin\javaw.exe (
Escribe el idioma. Puede ser aleman, frances, italiano o castellano: italiano
Meses en italiano
gennaio febbraio marzo aprile maggio giugno luglio agosto settembre ottobre novembre dicembre
Días de la semana en italiano, comenzando por el domingo
domenica lunedì martedì mercoledì giovedì venerdì sabato
```

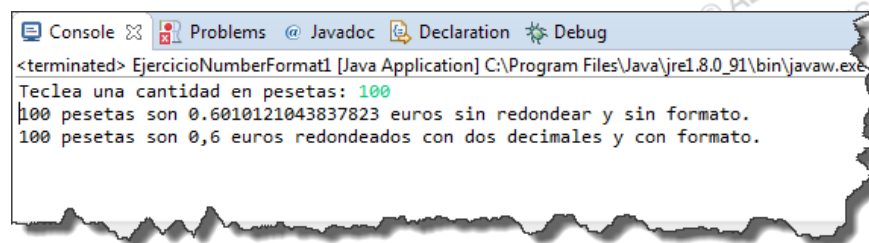
Ejercicio 2. Formatos numéricos

20

Realizar un programa que contenga una clase pública de nombre **EjercicioNumberFormat1** con un solo método: el main.

Solicitará al usuario un número que va a representar una cantidad en pesetas y las convertirá a euros, primero sin aplicar redondeo ni formato, y luego aplicando redondeo de dos decimales y el formato de número general asociado a un NumberFormat local

Datos a mostrar por consola



```
<terminated> EjercicioNumberFormat1 [Java Application] C:\Program Files\Java\jre1.8.0_91\bin\javaw.exe
Teclea una cantidad en pesetas: 100
100 pesetas son 0.6010121043837823 euros sin redondear y sin formato.
100 pesetas son 0,6 euros redondeados con dos decimales y con formato.
```

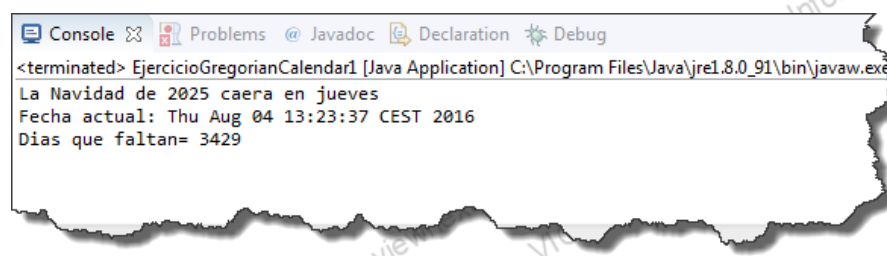
Ejercicio 3. GregorianCalendar

20

Realizar un programa que contenga una clase pública de nombre **EjercicioGregorianCalendar1** con un solo método: el main.

El programa debe averiguar en qué día de la semana caerá la Navidad del año 2025 y cuántos días quedan para que eso ocurra.

Datos a mostrar por consola



```
<terminated> EjercicioGregorianCalendar1 [Java Application] C:\Program Files\Java\jre1.8.0_91\bin\javaw.exe
La Navidad de 2025 caera en jueves
Fecha actual: Thu Aug 04 13:23:37 CEST 2016
Dias que faltan= 3429
```

Ejercicio 4. Cálculos con fechas.

35

Realizar un programa que contenga una clase de nombre **DiasVivosMejorado** que pedirá al usuario su nombre y su fecha de nacimiento en formato dd/mm/aaaa y le mostrará el día de la semana en que nació y los días que ha vivido hasta el instante de ejecución del programa.

Lo que se necesita para comenzar

```
package unidad10.ejercicios;
import java.text.*;
import java.util.*;
import java.io.*;

public class DiasVivosMejorado{
    public static void main(String args[]){
        DiasVivosMejorado dvm=new DiasVivosMejorado();
        try{
            dvm.comunicarUsuario();
        }catch(IllegalArgumentException iae){
            System.out.println("La fecha tecleada no existe o su formato es incorrecto");
            return;
        }
    }
    public void comunicarUsuario(){

    }

    private GregorianCalendar obtenerCalendar(String fechaNacString){

    }

    private void calcularDiaSemanaNacimiento(GregorianCalendar calNac){

    }

    private void calcularDiasVivos(GregorianCalendar calNac){

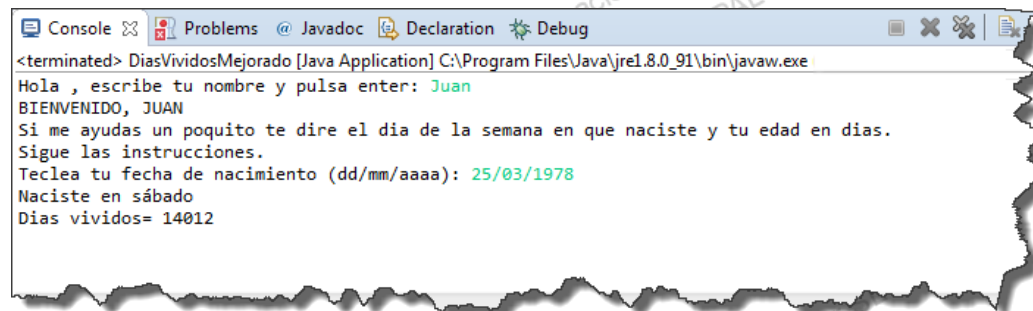
    }
}
```

Recomendaciones

- Los días de un mes deben pertenecer al intervalo [1,31], los meses a [1,12] y los años a [1,año actual]
- Considerar que no tiene sentido teclear fechas del estilo de 31/04/1980 pues Abril no tiene 31 días

- Pista: un buen método para evitar calendarios que admitan fechas incorrectas es `setLenient(boolean)` de `java.util.Calendar`
- Para comprobar el código, introducir la fecha actual. Los días vividos deberán ser 0

Datos a mostrar por consola



```
<terminated> DiasVividosMejorado [Java Application] C:\Program Files\Java\jre1.8.0_91\bin\javaw.exe
Hola , escribe tu nombre y pulsa enter: Juan
BIENVENIDO, JUAN
Si me ayudas un poquito te dire el dia de la semana en que naciste y tu edad en dias.
Sigue las instrucciones.
Teclea tu fecha de nacimiento (dd/mm/aaaa): 25/03/1978
Naciste en sábado
Dias vividos= 14012
```

Recursos

Enlaces de Interés



<http://docs.oracle.com/javase/tutorial/i18n/index.html>

<http://docs.oracle.com/javase/tutorial/i18n/index.html>

Formateo de fechas y números. Paquete java.text

Glosario.

- **Métodos deprecados:** Que un método esté deprecado significa que ya se ha creado otro que lo sustituye y que se utilizará en lugar del deprecado. Esta información aparece en el API de la clase que contiene el método.
- **Warnings:** Avisos generados por el compilador Java y que muestran posibles funcionamientos incorrectos de nuestro código. No sólo avisan de métodos deprecados.