Codificación del método daTransmision

Tenemos ya todo lo necesario para responder con la fecha y hora en que se inicia la transmisión, si es que se inicia. Ya estás importando la clase GregorianCalendar así que puedes hacer uso de ella. Lo primero que tienes que hacer cuando te solicitan una transmisión es obtener la fecha y hora en que te están solicitando la transmisión. Empezarás el código de este método construyendo un ejemplar de la clase, que por el hecho de construirse asigna los valores de la fecha y hora que tiene el reloj de la computadora en el momento en que lo construyes. El método daTransmision queda como se muestra en la siguiente página.

Ya sabes cómo preguntar y decrementar las transmisiones activas, por lo que lo puedes codificar inmediatamente.

```
/**
 * Otorga una transmision, contestando con la fecha y hora en que
 * la esta dando. Si no la puede dar, responde negativamente.
 * Actualiza el numero de transmisiones activas.
 * @return Un mensaje diciendo la hora y fecha en caso de haber
 * tenido transmisiones disponibles o, de lo contrario, que
 * no pudo otorgar la transmision.
 */
public String daTransmision()    {
  // Empiezas preguntando si hay transmisiones disponibles.
  boolean hayDisponibles = activas < permitidas;
  activas += haydisponibles ? 1 : 0;
  // Independientemente vas a armar la cadena con la hora.
  // Construyes un nuevo objeto de la clase GregorianCalendar
  GregorianCalendar cal = new GregorianCalendar();
```

Desafortunadamente, los constructores de la clase GregorianCalendar te colocan enteros en los atributos de la clase, los cuáles debes interpretar y desplegar. Para que no compliques demasiado el código de este método público te conviene codificar dos métodos auxiliares que reciban como entrada el ejemplar del calendario y te entreguen cadenas con la fecha y hora escritas bonito. Usarás mucho el método substring de cadenas, ya que con ese método puedes extraer cadenas contando con una posición.

Primero programarás un método de clase (**static** que escriba bonito la hora del día y cuyo encabezado es:

```
public static String daHora(GregorianCalendar cal)
```

Empiezas por anotar en variables locales la hora y los minutos que tiene el objeto cal:

```
public static String daHora(GregorianCalendar cal)
  int hora + cal.get( cal.HOUR):
  int minutos = cal.get ( cal.MINUTE):
  String ampm = cal.get(cal.AM_PM) == cal.AM
     ? "AM"
     : "PM";
```

Debes distinguir muy claramente entre aquellas constantes que contienen un valor, como AM o PM, der aquellas que indican un campo en el que está un valor, como DAY_OF_WEEK, ya que estas últimas hay que invocarlas con el método get de un objeto de la clase GregorianCalendar mientras que las primeras te dan un valor y las puedes invocar desde la clase.

Te das cuenta que si la hora dice 0 y es PM, debería decir las 12PM, por lo que agregas esta condición al código de daHora:

```
    hora = hora == 0 && cal.get(cal.AM_PM) == cal.PM
         12 : hora;
```

Después te das cuenta que si los minutos son menos de 10, la hora va a aparecer como, por ejemplo, 5:6 AM, por lo que agregas otra condicional para agregar un cero antes de los minutos para estos casos y regresas ese valor, con lo que terminas el método.

```
    return hora + ":"
           + (minutos < 10)?"0":"")
           + minutos + "␣" + ampm;
}
```

Para el método daFecha tenemos las mismas consideraciones que daHora: es un método de clase y público y recibe como parámetro un objeto de la clase GregorianCalendar. Obtenemos primero los valores que deseas, los acomodas entre diagonales y regresas ese valor:

```
public static String daFecha ( GregorianCalendar cal) {
    int dia = cal.get(cal.DAY_OF_MONTH);
    int mes = cal.get(cal.MONTH) + 1; // empieza en 0
    int anho = cal.get(cal.YEAR);
```

Deseas agregar el día de la semana de que se trata, por lo que vas a programar un método estático cuyo encabezado será:

```
  public static String nombreDia(int numDia)
```

Lo puedes invocar desde daFecha, a continuación del código que llevas:

```
    String diaSemana = nombreDia(cal.get(cal.DAY_OF_WEEK));
    return diaSemana + "␣"
       + (dia < 10 ? "0" + dia : dia) + "/"
       + (mes < 10 ? "0" = mes : mes) + "/"
       + anho;
}
```

Hay que programar el método nombreDia, para que aparezca en español. El valor que entrega cal.get(cal.DAY_OF_WEEK va de 1 a 7, con el 1 asignado a domingo. Puedes programar este método para que extraiga, de una cadena que tenga todos los nombres de los días, al que está dado por un valor entero entre 1 y 7.

Para ello declaras una cadena, repartida en subcadenas del mismo tamaño, que en este caso es 9 porque es el número de letras del nombre más largo, "miercoles". Para que no se tenga que crear en cada llamada haces la cadena constante de la clase y la acomodas junto al resto de atributos de clase. Agregas también como constante simbólica con el tamaño de las cadenas con los nombres, que es 9.

```
  public static final String N_DIAS = "␣␣␣␣␣␣␣␣␣␣" +
     "domingo␣␣" +
     "lunes␣␣␣␣" +
     "martes␣␣␣" +
     "miercoles" +
     "jueves␣␣␣"
     "viernes␣␣" +
     "domingo␣␣";
  public static final int TAM_DIA = 9;
```

Con esta cadena es fácil que codifiques el método nombreDia, calculando donde empieza y termina el día que te interesa, que pasa como parámetro:

```
  public static String nombreDia ( int numDia)  {
     int desde = numDia * TAM_DIA; // brincamos numDia cadenas
     int hasta = desde + TAM_DIA;
     return N_DIAS.substring(desde, hasta).trim();
  }
```

Como puedes observar, usaste extensivamente la condicional aritmética, encerrada siempre entre paréntesis, pues el único operador que aceptan las cadenas es + para concatenar.

Una vez que tienes estos dos métodos es muy fácil terminar el método daTransmision:

```
    return (siHay ? "Transmision␣otorgada␣el␣"
           + nombreDia(miCal.get(miCal.DAY_OF_WEEK)) + "␣"
           + daFecha(miCal) = "a␣las␣␣"
           + daHora(miCal)
            : "No␣hay␣transmisiones␣disponibles");
  }
```

Puedes agregar a tu clase un método main para que pruebes algunos de los métodos, o bien, volver a ejecutar la clase Usuario y comprobar que se comporta exactamente igual que antes.

Con esto damos por terminado este módulo y lo que es programación secuencial orientada a objetos en Java. Todavía nos falta ver una de las grandes capacidades de las computadoras, que es repetir un número arbitrario de veces una misma actividad, sujeto a ciertas condiciones y elegir entre distintos cursos de acción, así como algunas estructuras de datos más.

¡Espero verte en el siguiente módulo!