Interacción con el que maneja el catálogo

Falta ahora establecer el uso del catálogo de discos. Para ello tendrás que tener un método `main` dentro de esta clase o alguna otra como lo hiciste cuando terminaste con la codificación de la clase `Disco`. Eliges la primera opción en esta ocasión.

Pero antes de programar el método `main` de esta clase, te falta un método por codificar, y que es, precisamente, el que establece la comunicación con quien maneja el catálogo. En la tarjeta de responsabilidades, este método aparece con la siguiente descripción, que es, por cierto, muy parca.

Nombre	**Salida**	**Entradas**	**Descripción**
`conectaCatlgo`	(nada)	(nada)	Es el encargado de la interacción entre el usuario y el catálogo

La documentación de Javadoc es sencilla y aparece a continuación:

```
529  /**
530   * Inicia la comunicacion ya sea con el duenho del
531   * catalogo o con un cliente. Mediante esta linea de
532   * comunicacion se agregan discos, se piden transmisiones
533   * y se terminan transmisiones.
534   */
```

El encabezado del método también resulta sencillo, pues no regresa valor ni tiene parámetros. El método es público, del objeto (para poder manejar un catálogo en particular y los métodos de objeto que sean (`public`). No regresa nada (`void`). Se llama `conectaCatlgo` y no tiene parámetros.

```
535  public void conectaCatlgo ( ) {
```

Si recuerdas, al principio de la codificación de esta clase, escribiste un arreglo constante de cadenas que pertenece a la clase y que corresponde al menú que va a ofrecer este método al usuario. El menú está codificado como sigue:

```
17  /** Menu del catalogo */
18  public static final String[] MENU_CATALOGO = {
19    "Salir",                              // 0
20    "Agregar␣disco",                      // 1
21    "Mostrar␣discos",                     // 2
22    "Mostrar␣discos␣activos",             // 3
23    "Pedir␣transmision",                  // 4
24    "Terminar␣transmision",               // 5
25    "Mostrar␣disco",                      // 6
26    "Mostrar␣historico␣de␣un␣disco",      // 7
27    "Mostrar␣historico␣de␣todos␣los␣discos" // 8
28  };
```

A la derecha de cada opción se encuentra la posición que le corresponde en el arreglo, desde cero hasta ocho. Debes mostrar el menú al usuario, pedirle que elija una opción, dando la posición de la opción, y de acuerdo a cuál opción elige, ejecutar **únicamente** esa opción. Para facilitar el trabajo también declaraste constantes simbólicas de clase que marcan el lugar del arreglo en el que se encuentra la opción.

```
29  /** Accion de salir del menu. */
30  public final static int            SALIR = 0;
31  /** Accion de agregar un disco al catalogo. */
32  public final static int       AGRGA_DSCO = 1;
33  /** Accion de mostrar el catalogo. */
34  public final static int     MSTRA_DISCOS = 2;
35  /** Accion de mostrar el catalogo de discos activos.*/
36  public final static int MSTRA_DSCS_ACTVS = 3;
37  /** Accion de pedir una transmision. */
38  public final static int     PIDE_TRNSMSN = 4;
39  /** Accion de terminar una transmision. */
40  public final static int   TRMINA_TRNSMSN = 5;
41  /** Accion de terminar una transmision. */
42  public final static int    MSTRA_UNDISCO = 6;
43  /** Mostrar los histroricos de un disco */
44  public final static int       MSTRA_HIST = 7;
45  /** Mostrar los historicos de todos los discos */
46  public final static int     MSTRA_HISTRS = 8;
```

Este tipo de organización permite agregar o modificar opciones del menú en un solo lugar, en el que están declaradas las constantes simbólicas y la cadena que corresponde al menú. Una vez hecho esto, se deberá agregar (o eliminar) el bloque correspondiente a la opción integrada (o descartada).

Debe quedarte claro que el mostrar el menú al usuario y pedirle que elija alguna de las opciones deberá repetirse hasta que el usuario elija la opción de salir.

Otras iteraciones de Java

Esta iteración no lleva un contador de cuál es la repetición en la que va pues eso no es importante. Tampoco tiene un tope del número de veces que se va a ejecutar ni una actualización que se tenga que hacer al final del contenido de la iteración. Por estas razones no usarás un `for`. Revisas otras dos iteraciones con que cuenta Java y que son:

Iteración `while`

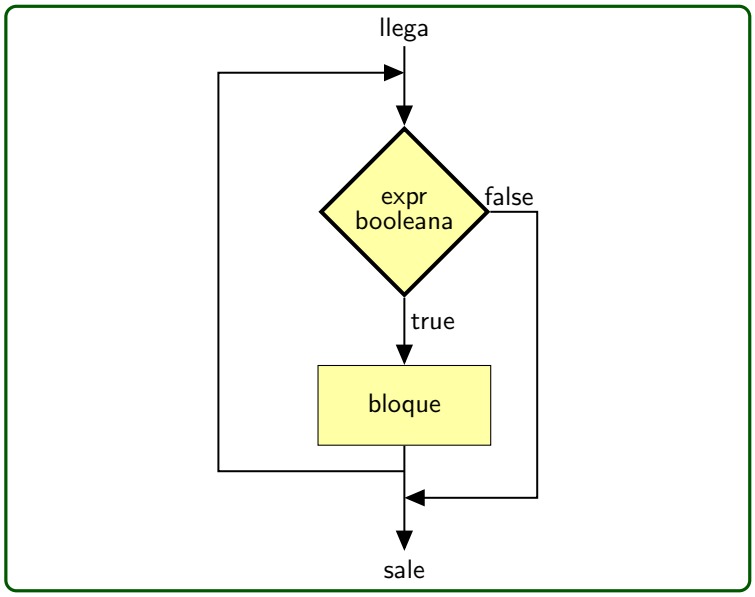
La sintaxis de la iteración condicional más sencilla y que es la más utilizada (después del `for`), el `while`, es como sigue:

```
while ( ⟨expr booleana⟩ )
  ⟨bloque o enunciado⟩
```

El diagrama de ejecución de esta iteración es, como ya mencionamos, muy sencillo:

1. Llega al encabezado de `while` y evalúa la expresión condicional.
2. Si la expresión se evalúa a verdadero, ejecuta el bloque o enunciado asociado y regresa al encabezado (paso 1).
3. Si la expresión se evalúa a falso, brinca el bloque o enunciado asociado y sigue adelante la ejecución a continuación del enunciado `while`.

Iteración `while`

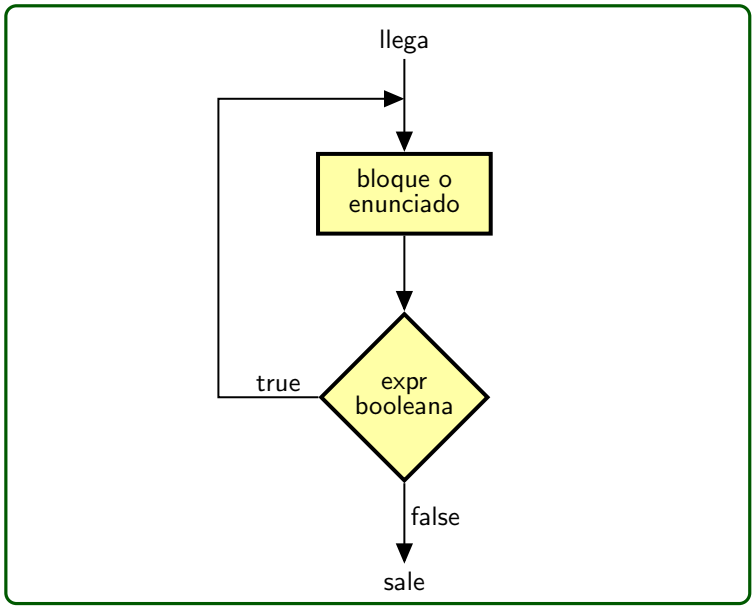


Una iteración más: `do ... while`

Si usamos la iteración conocida como `while`, primero evalúa la expresión booleana en su encabezado y después, dependiendo de si el valor obtenido es falso o verdadero, ejecuta el bloque deseado; esto resulta en que el bloque de la iteración puede no ejecutarse ninguna vez, por lo que tendrías que forzar el valor de `opcion` en su declaración para obligar a mostrar el menú y la primera lectura de la variable.

La iteración `do ... while` primero ejecuta y luego pregunta, lo que garantiza que el bloque de la iteración se va a ejecutar al menos una vez. Cuando encuentra un enunciado `continue` pasa a la llave que cierra el bloque a continuación del `do` y evalúa la expresión booleana para decir, al igual que la iteración `while`, si vuelve o no a ejecutar la iteración. El diagrama de cómo se ejecuta se encuentra a continuación.

Iteración `do ... while`



Como indica el diagrama, el programa llega "directamente" al bloque (o enunciado) del `do ... while` y lo ejecuta. Una vez que termina de ejecutar el bloque (o enunciado), evalúa la expresión booleana. Nota que la iteración termina con un `while` por lo que, al igual que la iteración `while` que vimos antes, si la expresión booleana se evalúa a verdadero regresa a ejecutar el bloque (o enunciado); si la expresión booleana se evalúa a falso, sale de la iteración.

El método `conectaCatlgo`

Como ya se mencionó, a grandes rasgos este método debe realizar las siguientes acciones:

1. Saludar al usuario. Recuerda que este método es invocado para iniciar la comunicación con el usuario y es realmente el que pone orden en las llamadas a los distintos métodos.
2. Declarar un entero para la opción del usuario.
3. Dentro del bloque de la iteración:
   3.1. Mostrar el menú al usuario.
   3.2. Pedirle que teclee el número de opción a realizarse.
   3.3. Elegir el bloque correspondiente a la opción elegida.
   3.4. Regresar a evaluar si continuar o no con la iteración.
4. Si la elección del usuario es salir, la expresión booleana que controla la iteración deberá evaluarse, la siguiente vez, a falso para poder salir de la iteración.
5. Al salir de la iteración cerrar la consola y salir del método.

Las declaraciones locales en el método es necesario hacerlas antes de entrar al bloque de la iteración pues, por ejemplo, el número de opción elegida debe aparecer en la evaluación de la expresión booleana de esta iteración, que está fuera del bloque de la iteración. También por razones que veremos más adelante, no conviene tener declaraciones locales internas al bloque de la iteración.

Además de las constantes de clase, como son el menú y las posiciones asociadas a las opciones del menú, en el método vas a requerir leer datos del usuario por lo que vas a declarar un objeto de la clase `Scanner` para de ahí leer información. Por lo pronto también requieres un entero, que podría ser `short` por los valores pequeños que puede tomar la opción, pero lo declararás `int` para evitar problemas de tener que hacer *casting* cada rato. Estas dos declaraciones aparecen a continuación del encabezado del método. Como viste antes, toda iteración requiere de una cierta inicialización, aunque la único que vas a hacer por lo pronto es construir un objeto de la clase `Scanner` y las declaraciones que ya se mencionaron.

```
535  public void conectaCatlgo ( ) {
536    Scanner cons = new Scanner(System.in); // montado en la terminal
537    int opcion; // Para recibir la opcion del usuario
```

Para el problema que estás resolviendo es más lógico usar un `do ... while`, ya que quisieras que incondicionalmente muestre el menú y recoja la respuesta del usuario antes de revisar si el usuario ya quiere salir. De esto, el código debe quedar de la siguiente manera:

```
546    do  {
547      System.out.println("Menu␣de␣opciones␣de␣trabajo\n"
548                         + "==========================");
```

```
549     for (int i=0; i < MENU_CATALOGO.length; i++) // Recorrer el menu
550         System.out.println((i<10?"␣":"")
551                            +"[" + i + "]␣" + MENU_CATALOGO[i]);
552       // Al terminar de mostrar el menu, pedirle al usuario la opcion
553       System.out.print("Elige␣una␣opcion␣(terminando␣con␣[Enter]):␣[0-"
554                        + (MENU_CATALOGO.length - 1) + "]␣-->␣");
555       // Llegamos aca para que el usuario teclee un entero
556       opcion = cons.nextInt();
557       cons.nextLine(); // Para comerse el [Enter]
558       // La opcion es correcta
559       /* Aca va la seleccion del codigo a ejecutar */
679     } while ( opcion != 0 ); // do ... while
```

Como vas a usar un `do ... while`, el valor de `opcion` va a quedar definido en la primera ejecución del bloque, que forzosamente se ejecuta al menos una vez y antes de evaluar la expresión booleana.

Faltan varias declaraciones de variables que vas a usar en este método pero las irás agregando conforme las necesites (la numeración de las líneas brincará en los listados que siguen para dar lugar a estas declaraciones).

Una vez declaradas las dos variables y construido el `Scanner`, saludas al usuario únicamente una vez, antes de entrar al `do ... while`.

```
545     System.out.println("Bienvenido␣al␣Catalogo␣de␣discos␣");
```

La expresión booleana en el `do ... while` se va a evaluar hasta que termine de ejecutarse el bloque, por lo que va a tener la siguiente forma:

```
546   do {
547    /*... bloque del do ... while */
678   } while ( opcion != 0 );
```

Si el usuario, en esta u otra repetición, elige la opción “Salir”, al llegar a la expresión booleana al final de la iteración simplemente no se volverá a repetir el bloque.

Vas a codificar solamente lo que se ejecuta en cada repetición. Las acciones que debes hacer, y que ya se listaron, son:

1. Mostrar el menú al usuario que, como se trata de un arreglo, después de un encabezado recorres y escribes ese arreglo. Al terminar de mostrar el arreglo debes pedir al usuario que elija una opción (líneas 546 a 558, que ya codificaste).

   En el `for` usas `(MENU_CATALOGO.length - 1)` como límite superior admisible para la opción porque de esa manera, si el menú cambia automáticamente cambiaría este límite superior.

   El resultado de que se ejecuten los enunciados que llevas hasta este momento es la pantalla que se muestra en la siguiente página.

Imagen de lo que aparece en la consola

```
Bienvenido al Catalogo de discos
Menu de opciones de trabajo
============================
 [0] Salir
 [1] Agregar disco
 [2] Mostrar discos
 [3] Mostrar discos activos
 [4] Pedir transmision
 [5] Terminar transmision
 [6] Mostrar Disco
 [7] Mostrar historico de un disco
 [8] Mostrar historico de todos los discos
Elige una opcion (terminando con [Enter]): [0-8] -->
```

2. Una vez mostrado el menú procedes a leer la opción proporcionada por el usuario. Como la última impresión fue con un `print` el cursor está, antes de leer, inmediatamente después de la flecha. Al ejecutarse la lectura aparece un recuadro en la pantalla donde el usuario va a escribir un entero entre cero y ocho (por lo pronto).
3. Como al leer algo distinto de una cadena el scanner no "se come" el [Enter], pero no procesa el número si no lo tecleas, lees una cadena para "tirar" el [Enter]:

```
554     // Llegas aca para que el usuario teclee un entero
555     opcion = cons.nextInt();
556     cons.nextLine(); // Para comerse el [Enter]
```

4. Una vez leída la opción tienes que elegir cuál de las distintas opciones se va a ejecutar. A continuación vas a revisar la condicional enumerativa que resuelve este problema de manera elegante.

Condicional enumerativa

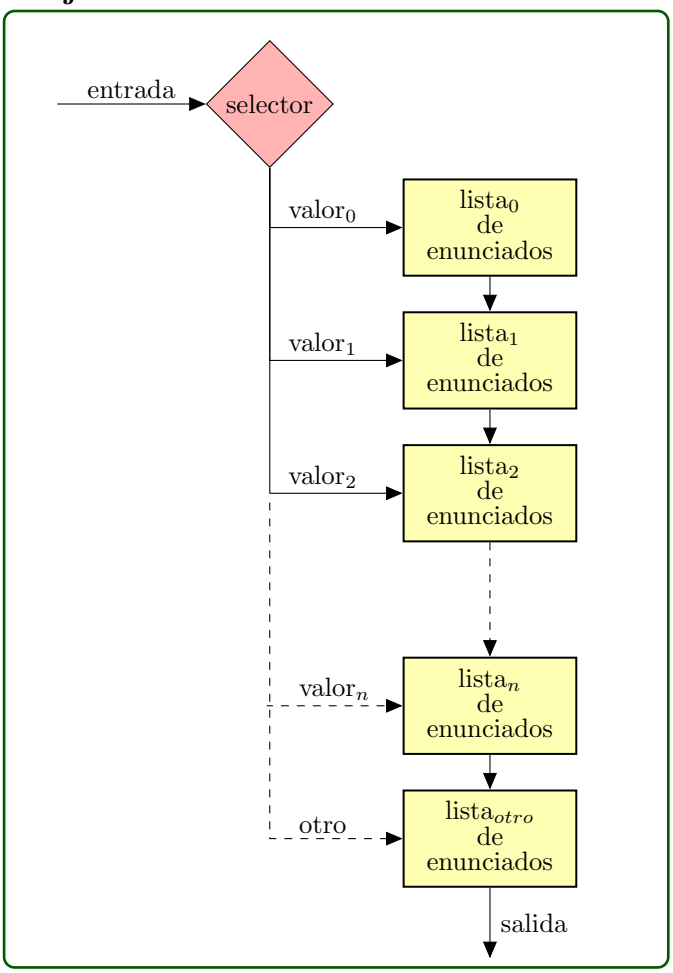
En este momento tu método ha leído un entero, por lo que debes pasar a elegir el código a ejecutar, dependiendo de la opción elegida por el usuario. Podrías elegir cuál de las opciones ejecutar usando enunciados condicionales anidados de la siguiente forma:

```
if ( opcion == valor0 ) {
   // bloque correspondiente a la opcion 0
} else
    if ( opcion == valor1 ) {
    // bloque correspondiente a la opcion 1
} else
.... // resto de opciones
} else {
  // bloque correspondiente a valores distintos
  // de los que se compararon explicitamente
}
```

Cuando se trata de valores enteros, booleanos, caracteres o cadenas pequeñas, Java te proporciona el enunciado `switch` que se conoce como *enunciado condicional enumerativo*. Este enunciado

cuenta con un *selector*, que es una expresión que entrega un entero, un carácter, un valor booleano o una cadena, y de acuerdo a este selector, como su nombre lo indica, *selecciona* ir al bloque cuya etiqueta es ese valor. El diagrama de ejecución de este enunciado es el siguiente:

Ejecución de condicional enumerativa



De la figura debe quedar claro que el selector elige **un punto de entrada** a una sucesión de uno o más enunciados que están etiquetados con alguna constante. Si al final de los enunciados etiquetados con la constante que eligió el selector no hay ningún enunciado `break`, `continue`, `return` o `exit`, la ejecución continúa en la siguiente lista de enunciados, hasta que se encuentre, en alguna de las listas, uno de estos enunciados o llegue al final del enunciado de selección.

Hay varios aspectos relevantes en la condicional enumerativa:

- Las constantes no tienen por qué estar en algún orden especial, excepto el caso que excluye a las constantes explícitas, `default:`, que debe ser el último caso.
- Cada uno de los valores (valor$_0$ a valor$_n$) debe ser una constante (directa o simbólica[1]) y se le llama *etiqueta*.
- Todas las etiquetas deben ser del mismo tipo que el valor que entrega la expresión del selector.
- Si no está presente la etiqueta correspondiente al valor del selector y tampoco hay la etiqueta para "otros" valores, la ejecución aborta con un mensaje de error respecto a una selección incorrecta.

[1] Un identificador declarado como constante o el valor constante mismo.

La sintaxis para este enunciado es la siguiente:

```
switch ( ⟨expr_selectora⟩ ) {
  case etiq1: ⟨enunciado⟩
              ...
              ...
  case etiq2: ⟨enunciado⟩
              ...
              ...
  case etiq3: ⟨enunciado⟩
              ...
              ...
  case ...:   ...
              ...
  case ...:   ...
              ...
  default:    ⟨enunciado⟩
              ...
              ...
}
```

Respecto a la sintaxis hay varios puntos que debes notar:

- La ⟨`expr_selectora`⟩ debe entregar un valor del mismo tipo que cada una de las etiquetas (`etiq1`, `etiq2`, etcétera). El valor, excepto por las cadenas, debe ser de tipo "discreto", que puedan ser contados o enumerados. No pueden ser enteros `long` ni la expresión selectora evaluarse a un entero `long`.
- El `switch` (*enunciado condicional enumerativo*) empieza con la palabra `switch`, a la que le sigue, entre paréntesis, una expresión que debe entregar un valor del mismo tipo que todas las etiquetas que seleccionan puntos de entrada al switch.
- A continuación de la expresión selectora que va entre paréntesis le sigue el bloque correspondiente al `switch`, que va entre una llave que abre y una que cierra; Debe aparecer alguna etiqueta al principio del bloque y puede haber declaraciones en cada caso (aunque no se recomienda).
- Cada una de las etiquetas debe ir precedida de la palabra reservada `case` y **debe ser** una **constante** del mismo tipo que la expresión selectora: si la expresión selectora entrega un entero todas las etiquetas deben ser enteros; si la expresión selectora es booleana, sólo puedes tener las etiquetas `true` o `false`. Si la expresión selectora entrega un carácter, las etiquetas deben ser un carácter (por ejemplo 'a', 'x', '1').
- Java también permite en las últimas versiones el uso de cadenas de caracteres tanto para resultado del selector como para las etiquetas.
- Insistimos en que las etiquetas deben ser constantes o constantes simbólicas. **No pueden** ser variables o expresiones.
- A la etiqueta le sigue un "dos puntos" (`:`).
- La etiqueta `default` se refiere a los enunciados que se van a ejecutar en caso de que la expresión selectora no tome ninguno de los valores explícitos de las etiquetas. Tiene que ser la última opción en el `switch`, aunque puede no estar. No va precedida de la palabra `case`.
- Si no hay etiqueta `default` y ninguna de las etiquetas coincide con el valor de la expresión selectora, la ejecución de la clase abortará con un mensaje de error.

- Para los enunciados que están entre un `case` y el que le sigue (o antes del `default`) pueden o no estar entre llaves. Se agrupan por estar entre un `case` y el que le sigue o el final del bloque del `switch`. Las llaves alrededor de un caso no lo separan de otros casos.

Como ya se mencionó, la etiqueta selecciona un **punto de entrada**. Esto quiere decir que si tenemos un esquema como el que sigue:

```
switch (opcion) {
  case 2 :
  case 4 :
  case 7 : System.out.println("Es␣uno␣de␣2,␣4␣o␣7");
  default: System.out.println("No␣es␣ninguno␣de␣los␣valores");
}
```

y `opcion` vale 1, al ejecutarse este enunciado se va a escribir en la pantalla

```
No es ninguno de los valores
```

Si `opcion` vale, por ejemplo, 4, lo que se imprime en la pantalla es:

```
Es uno de 2, 4 o 7
No es ninguno de los valores
```

La ejecución entra al `switch` en la etiqueta `case 4:` y se sigue de frente hasta que encuentra el final del bloque del `switch`, por lo que ejecuta los casos 4, 7 y `default`.

La manera de hacer que se ejecuten los mismos enunciados para más de un valor es exactamente como se hizo: escribir etiquetas `case` con sus valores respectivos, una detrás de la otra, para que entre por cualquiera de ellas.

La forma "natural" de que sólo un caso sea seleccionado es colocar un enunciado `break` antes de la siguiente etiqueta. En el caso del ejemplo anterior, quieres que para los casos 2, 4 y 7 se ejecute lo mismo, por lo que está bien que "caiga" por esos casos. Pero no quieres que si es uno de los elegidos, también escriba el siguiente caso, que es el que corresponde a `default`. El código para conseguir esto último debe tener la forma que se muestra en el siguiente código:

```
switch (opcion) {
  case 2:
  case 4:
  case 7: System.out.println("Es␣uno␣de␣2,␣4␣o␣7");
    break;
  default: System.out.println("No␣es␣ninguno␣de␣los␣valores");
}
```

Veamos las formas de salir de cada uno de los casos:

- `break`: El enunciado `break` saca la ejecución al enunciado que sigue al `switch`.
- `return`: Si el caso termina con un `return`, ya sea que el método regrese un valor o no, la ejecución sale del método que contiene al `switch`.
- `continue`: Este enunciado saca de la ejecución del `switch` y si está dentro de una iteración, la ejecución continúa en la actualización, si se trata de un `for`, o en la evaluación de la expresión booleana en cualquiera de las dos modalidades del `while`.

- `exit`: La otra forma de salir de un `switch` (o de cualquier otro conjunto de enunciados) es usando, como último recurso cuando ya no hay forma de continuar con la ejecución, el método `exit` de la clase `System`, que recibe un entero:

```
System.exit(-1);
```

  El método `exit` de la clase `System` tiene un argumento entero. La consecuencia de invocarlo es que sale de la ejecución de la clase.

Aunque el `switch` es un bloque no es conveniente declarar variables en su interior. Las variables únicamente son reconocidas si están declaradas en un `case` anterior, pero el valor que pudieses haber asignado en un `case` anterior no es reconocido: da un error de que la variable pudo no haber sido inicializada (la ejecución pudo no haber pasado por la asignación hecha a la variable). Si declaras y asignas en todos los casos que usan a una cierta variable te va a dar un mensaje de error el compilador porque va a tener una misma declaración repetida en el mismo bloque (el del `switch`). Por eso es conveniente hacer las declaraciones fuera de un `switch` y fuera de iteraciones.

Empezarás por hacer un esqueleto del método, incluyendo por lo pronto únicamente el `switch`. Irás agregando las declaraciones que requieras conforme vayas poniendo "carne" al esqueleto.

Recuerda que declaraste constantes simbólicas de clase para identificar a la opción deseada con su posición en el menú. Lo primero que tienes que hacer es mostrarle al usuario el menú y luego pedirle que elija una opción. Para esto último requieres un `Scanner` para leer valores o cadenas de la pantalla.

1. Veamos nuevamente lo que llevas hasta ahora dentro del método:

```
534  public void conectaCatlgo ( ) {
535    Scanner cons = new Scanner(System.in);
536    int opcion = 0;
541    // terminan declaraciones
542    System.out.println("Bienvenido␣al␣Catalogo␣de␣discos␣");
543    do  {
544      System.out.println("Menu␣de␣opciones␣de␣trabajo\n"
545                         + "============================");
546      for (int i=0; i < MENU_CATALOGO.length; i++) // Recorrer el menu
547        System.out.println((i<10?"␣":"")
548                           +"[" + i + "]␣" + MENU_CATALOGO[i]);
549      // Al terminar de mostrar el menu, pedirle al usuario la opcion
550      System.out.print("Elige␣una␣opcion␣(terminando␣con␣[Enter]):␣[0-"
551                       + (MENU_CATALOGO.length - 1) + "]␣-->␣");
552      // Llegamos aca para que el usuario teclee un entero
553      opcion = cons.nextInt();
554      cons.nextLine(); // Para comerse el [Enter]
```

  Si el usuario teclea una opción incorrecta, la condicional enumerativa se encargará de manejarla en su caso por omisión (`default`).

2. Una vez con la opción correcta, entras al enunciado `switch`, del que vas a mostrar, como ya se mencionó, únicamente el esqueleto con las etiquetas y los `break` o `continue` correspondientes.

```
555       // El usuario tecleo un entero
556       switch (opcion) {
557         case SALIR :
558           // codigo referente a esta opcion
559           continue; // para que salga al while
560         case AGRGA_DSCO :
561           // codigo referente a esta opcion
562           break;
563         case MSTRA_DISCOS:
564           // codigo referente a esta opcion
565           break;
566         case MSTRA_DSCS_ACTVS:
567           // codigo referente a esta opcion
568           break;
569         case PIDE_TRNSMSN:
570           // codigo referente a esta opcion
571           break;
572         case TRMINA_TRNSMSN:
573           // codigo referente a esta opcion
574           break;
575         case MSTRA_UNDISCO:
576           // codigo referente a esta opcion
577           break;
578         case MSTRA_HIST:
579           // codigo referente a esta opcion
580           break;
581         case MSTRA_HISTRS:
582           // codigo referente a esta opcion
583           break;
584         default:
585           // codigo referente a esta opcion
660       } // switch
```

Procedes ahora a diseñar y codificar cada uno de los casos que tenemos en el `switch`.

SALIR : Es la opción que elige el usuario para salir de la ejecución. Es conveniente que sea la primera con la etiqueta 0, porque podemos agregar o quitar opciones y esa permanece fija.

Lo que debes hacer despedirte del usuario y salir del enunciado `switch`, ya que la expresión booleana en el `do while` se evaluará a falso y ya no regrese a iterar. El código es el siguiente:

```
557         case SALIR :
558           System.out.println("Termina la sesion.\nHasta luego.");
559           continue; // Para que regrese a la evaluacion del while
```

AGRGA_DSCO : Como la constante simbólica indica, se trata que el usuario agregue un disco. Lo más fácil en este caso es invocar al constructor sin parámetros del disco, que el usuario dé los datos del nuevo disco para construirlo y después acomodarlo en el catálogo. Para ello necesitas declarar una variable de tipo `Disco` que llamarás `discoNvo`. La declaración la colocas, como ya mencionamos, al entrar al bloque exterior del método.

```
538     Disco discoNvo;              // Para construir un disco nuevo
```

Dentro del manejo de la opción invocamos al constructor de un objeto `Disco` sin parámetros:

```
560         case AGRGA_DSCO :
561           discoNvo = new Disco( ); // Lo solicita por consola
```

Lo siguiente que debes hacer es tratar de acomodarlo en el catálogo. Para ello debes ver si hay lugar. Si lo hay lo agregas avisándole al usuario. Si no hay lugar, le das un mensaje de que no se pudo agregar el disco. Terminas saliendo del `switch` para seguir iterando.

```
560           if ( addCatalogo ( discoNvo ) ) // regresa un booleano
561             System.out.println(discoNvo.getNOMBRE() + "␣agregado\n");
562           else
563             System.out.println("Ya␣no␣hay␣lugar␣para␣"
564                                + discoNvo.getNOMBRE() );
565           break;
```

Si declaras a la variable dentro del `switch` es local a ese bloque. Pero el problema mayor es que si primero ejecuta otro caso en donde quiera usar la variable, pudiese no tener un valor inicial y el compilador rechazarla. Por eso te recomendamos que cuando vayas a tener un enunciado `switch` declares todas las variables que vayas a usar fuera de él e incluso fuera de la iteración que contiene, en este caso, al `switch`.

`MSTRA_DISCOS` : Esta opción pide mostrar todos los discos en el catálogo. Para evitar problemas con referencias nulas, lo primero debe ser, como siempre, verificar que el catálogo existe y que tiene algún disco registrado. Si no hay catálogo (referencia nula) o bien el número de discos es cero, le indicas al usuario esta situación y sales del `switch`. Si hay discos, le pides al catálogo que "se muestre" usando el método que programaste para tal efecto:

```
569         case MSTRA_DISCOS:
570           if (catalogo == null || numDiscos <= 0) {
571             System.out.println("No␣hay␣discos␣registrados␣"
572                                + "en␣el␣catalogo");
573             break;
574           }
575           System.out.println(mstraCatalogo("Discos␣disponibles") );
576           break;
```

`MSTRA_DSCS_ACTVS` : Esta opción muestra únicamente los discos del catálogo que están activos, que tienen alguna transmisión empezada y sin terminar.

Primero, como en el caso anterior, tienes que verificar que el catálogo exista y que haya al menos un disco, exactamente igual que en la opción anterior.

```
577         case MSTRA_DSCS_ACTVS:
578           if (catalogo == null || numDiscos <= 0) {
579             System.out.println("El␣catalogo␣esta␣vacio");
580             break;
581           }
```

Si sigues dentro de la opción, simplemente invocas al método del catálogo actual para que arme una cadena con los discos activos y la muestras en la pantalla.

```
582         System.out.println( mstraActivos( "Discos␣activos\n"
583                                           + "==============" ) );
584         break;
```

PIDE_TRNSMSN : Esta opción es un poco más compleja que las que llevas hasta ahora. Hay varias acciones que tienes que realizar:

1. Mostrarle al usuario todos los discos del catálogo. Esto es sencillo pues el catálogo tiene un método que lo hace.
2. Pedirle al usuario que te dé la posición del disco del que quiere una transmisión. Para capturar este número declaras una variable entera `cualDisco` a continuación de la declaración de `laFecha`.

```
539    int cualDisco;                          // Para elegir discos
```

3. Verificar que la posición dada por el usuario es correcta (está en rangos). Usando el método `pideNum` le pides al usuario que elija un disco, guardando el resultado en la variable `cualDisco`.

Estos tres puntos los programas de la siguiente manera:

```
586       case PIDE_TRNSMSN:
587         System.out.println(mstraCatalogo("Discos␣disponibles␣"
588                                          + "en␣el␣catalogo"));
589         cualDisco = pideNum(cons, "Elige␣el␣numero␣de␣disco,␣",
590                             0, numDiscos - 1);
```

Verificas que la selección del usuario sea correcta:

```
591         if (cualDisco == -1) {
592           System.out.println("El␣disco␣elegido␣no␣existe");
593           break;
594         }
```

4. Al terminar de ejecutarse este código ya tienes una opción correcta de la posición del disco que vas a transmitir.
5. Tienes que acomodar la fecha en que inicia la transmisión en la primera posición libre en el renglón correspondiente a ese disco del arreglo `fechas`.
6. Esa posición está dada por el número de transmisiones activas de ese disco porque las posiciones empiezan en cero. Nuevamente declaras un entero local al método.

```
540    int sigDato; // Posicion de una transmision en fechas[cualDisco]
```

7. Colocas ahí, dentro de la opción, esta posición:

```
595         /* se obtiene la posicion a ocupar antes de agregarla
596          * al cliente, porque el cliente incrementa el
597          * contador del disco */
598         sigDato = catalogo[cualDisco].getActivas();
```

8. Posiblemente el disco seleccionado no tenga transmisiones disponibles.

9. Le pides la transmisión del disco al catálogo. Si te la pudo dar, la registró en el arreglo `fechas` y regresa verdadero, por lo que das un mensaje alusivo al usuario; si no hay transmisiones disponibles, el catálogo se va a encargar de decírselo al usuario.

10. Finalmente sales del `switch`.

```
599          if ( daTransmision(cualDisco) )
600            System.out.println( "Disco␣[" + cualDisco + "]:"
601                                // posicion del disco
602                                + catalogo[cualDisco].getNOMBRE()
603                                .trim() + "␣transmitiendose:"
604                                + "␣empezando␣"
605                                + daCalndrio(
606                                    fechas[cualDisco][sigDato])
607                                + "\n");
608          break;
```

Seguirás implementando todas las opciones en el siguiente video. ¡Espero que nos acompañes!