Programación II

Curso 24-25

Práctica 2

0 Cuestiones previas

Los programas que escribas deben respetar un correcto estilo de programación, sangrando adecuadamente el código fuente, eligiendo identificadores significativos, y añadiendo comentarios apropiados para explicar lo que haces.

Crea el directorio p2 dentro del directorio `Practicas`. En ese directorio guardarás todos los ficheros fuente de esta práctica, y en él crearás todos los ficheros `.class`.

1 Actividad 1

Antes de realizar la siguiente actividad deberías haber leído el documento previo asociado a la práctica anterior ("**Ficheros**"): revisa la descripción de cómo leer de un fichero de texto (sección 3).

1.1 Tarea 1a

Escribe el programa `ReadCoworkersFile.java` que hace lo siguiente:

1. Recibe un argumento que representa el nombre de un fichero con información sobre cotrabajadores.
2. Abre ese fichero y lee sus distintas líneas. Si son comentarios, las descarta.
3. Divide cada línea que represente a un cotrabajador en sus distintos campos (recuerda el método `split`), y los imprime en pantalla siguiendo el siguiente formato dependiendo de los campos que haya:

```
Tipo=T  ID=I  Nom.=N  Sen.=S  Cou.=C  CCard=CC  Com.=CO  BAcc.=BA  PId=PI  Fechas=F1-F2…
```

Por ejemplo, para la línea

```
"P;11118888X;David;1,1;Spain;1111-2222-3333-4444;02/04/2024;30/11/2024"
```

la salida sería

```
Tipo=P  ID=11118888X  Nom.=David  Sen.=1,1  Cou.=Spain  CCard=1111-2222-3333-4444
Fechas=02/04/2024-30/11/2024
```

Guarda el programa, compílalo, ejecútalo, y comprueba que hace lo que se pide.

1.2 Tarea 1b

Escribe el programa `ReadWriteCoworkersFile.java` que hace lo siguiente:

1. Recibe dos argumentos que representan nombres de ficheros.
2. Abre el primer fichero y lee sus distintas líneas. Descarta los comentarios.
3. Divide cada línea que represente a un cotrabajador en sus distintos campos, y los almacena en un array bidimensional (una fila por cotrabajador).

Pista. Para decidir el número de filas del array hay que saber antes el número de cotrabajadores. Una opción es leer dos veces el fichero: tras la primera sabes el número de cotrabajadores (N) y puedes crear el array de filas:

```
String[][] coworkers = new String [N][];
```

Pista. El número de columnas de cada fila no es fijo (depende del tipo de cotrabajador y de las fechas de sus anteriores estancias), así que deberás ir creando los distintos arrays sobre la marcha, tras leer cada línea. Ten en cuenta que el método `split` ya crea y devuelve un objeto array:

```
bucle que lee las N líneas:
        leer la linea_j
        String[] componentes = linea_j.split(";"); // crea un array
        coworkers[j] = componentes;
        // o directamente coworkers[j] = linea_j.split(";");
```

4. Guarda la información del array bidimensional en el segundo fichero, en el formato conocido, pero intercambiando las columnas de ID y nombre. Si un empleado (sea o no ejecutivo) tiene una antigüedad de más de 10 meses, se imprimirá después de su línea otra línea con el comentario "# long seniority" (para comprobarlo deberás convertir lo leído del fichero a *float*, por ejemplo, usando `Float.valueOf()`).

   Atención. Fíjate en el separador de decimales del fichero. ¿Qué espera recibir `Float.valueOf()`? Recuerda el método `replace()` de la clase `String`.

Guarda el programa, compílalo, ejecútalo, y comprueba que hace lo que se pide.

1.3 Tarea 1c

Crea el paquete p2. Modifica todos los ficheros de esta práctica para que pertenezcan a este paquete (lee antes el documento previo asociado a la práctica -"**Paquetes**"-).

Vuelve a compilar y ejecutar todos los programas de esta práctica para ver que todo funciona correctamente. Hazlo desde el directorio padre del paquete, pero también desde otro directorio cualquiera, especificando antes el CLASSPATH o usando ***-cp***.

2 Actividad 2: crear la clase `Particular`

Siguiendo el esquema de la información presentada hasta este momento, un cotrabajador particular lo caracterizamos por los siguientes atributos básicos (**a los que no se podrá acceder directamente desde el exterior de la clase**):

- **id**: una cadena de caracteres representando el DNI, según el formato "DDDDDDDDL", es decir, compuesta por 8 dígitos y una letra.
- **name**: una cadena de caracteres compuesta sólo por letras.
- **seniority**: un número real positivo (*float*) menor o igual a 30.
- **country**: una cadena de caracteres compuesta de máximo 20 letras.
- **creditCard**: una cadena de caracteres según el formato DDDD-DDDD-DDDD-DDDD, o sea, cuatro grupos de cuatro dígitos, separados por guiones.

Para representar un cotrabajador particular, creamos la clase `Particular`, parcialmente definida en el siguiente cuadro:

```
public class Particular {

    // Atributos de instancia (completar con el resto de atributos de instancia)
    … id; // completar visibilidad y tipo, no accesible directamente desde el exterior de la clase

    // Atributos estáticos (completar con el resto de atributos estáticos, las distintas constantes)
    …  ID_FORMAT = "DDDDDDDDL";       // D=0..9, L=A..Z, constante accesible desde el exterior de la clase

    // 3 constructores: vacío, con argumentos, y de copia
    public Particular ()  {
    }
    public Particular (… id, … name, … seniority, … country, … creditCard) {      // completar tipos
          this.id = id;   // completar el código
    }
    public Particular (Particular p) {…
    }

    // Métodos. Completar tipos y visibilidad para que sean accesibles desde el exterior de la clase

    // Métodos estáticos o de clase (completar con el resto de métodos estáticos, usando las constantes)
    … boolean isValidId (… id) {     // completar la visibilidad y los tipos
                    … // completar el código
    }

    // Métodos de instancia (completar getters, setters y toString)
    …  String  toString () {
         // devolver una cadena conforme al formato de representación conocido  ("P;I;N;S;C;CC")
    }
}
```

Todos los métodos (de instancia y de clase) serán accesibles desde el exterior de la clase.

Copia el código anterior y complétalo (en los puntos suspensivos), según se indica:

- Completa los atributos de instancia y estáticos con su tipo y visibilidad correcta.
- Completa el código del constructor con argumentos y el del método de clase `isValidId()`.
- Crea el resto de métodos estáticos con su código de validación correspondiente.
- Completa los *getters*, *setters* y el método `toString()`. En el caso de `toString()`, escribe la antigüedad siempre con dos cifras decimales usando `String.format(".2f"…)`.

3 Actividad 3: crear y manipular objetos de la clase `Particular`

Copia la clase P2, parcialmente definida en el siguiente cuadro, que contiene el método `main`.

```
public class P2 {
  public static void main(String[] args)  {
    // atributos de varios cotrabajadores particulares
    String id3="11113333C",  id4="2222444D",    id5="33335555E",  id6="40005000X";
    String name3="Alex",     name4="David",     name5="Oscar0",   name6="Leo";
    float  sen3 =2.6F,       sen4 = 31.8F,      sen5=1F,          sen6=-3.0F;
    String country3="Spain", country4="France", country5="",      country6="Italy";
    String crd3="1111222233334444", crd4="2222-3333-4444-5555", crd5="0000 3333 2222 1111", crd6="3333-2222-1111-0000";
  }
}
```

A partir de esta clase, crea código en el método `main` para lo siguiente:

1. Usando el constructor con argumentos de la clase `Particular`, crea un objeto con los siguientes valores: *id*="11112222A", *name*="David", *seniority*=1.8, *country*="Spain", *creditCard*="1234-4321-5678-8765".
2. Crea otro objeto `Particular` usando el constructor vacío y, con los *setters*, dale estos valores a sus atributos: *id*="33334444B", *name*="Sophia", *seniority*=2.7, *country*="France", *creditCard*="0123-3210-4444-5555".
3. Crea otro objeto `Particular` con el constructor de copia a partir del anterior.
4. Usando el método `toString()`, saca por pantalla los datos de los cotrabajadores particulares creados.
5. Usando cualquiera de los constructores que desees, crea objetos según los datos para los cotrabajadores 3, 4, 5 y 6 detallados en el cuadro anterior, siempre y cuando todos los valores de sus atributos sean correctos. Para comprobar si son correctos, usa (antes de crear los objetos) los atributos y métodos estáticos de la clase. Por ejemplo:

   a. Si el valor del atributo *id* es incorrecto, saca el siguiente mensaje por pantalla:

      "Incorrect id value: <ID>. Not comply with format <ID_FORMAT>"

      donde <ID> representa el valor de *id* incorrecto e <ID_FORMAT> el formato obtenido de la correspondiente constante de la clase.

   b. Si el valor del atributo *name* es incorrecto, saca el siguiente mensaje por pantalla:

      "Incorrect name value: <NAME>. Valid name must contain only letters"

      donde <NAME> representa el valor de *name* incorrecto.

   c. Si el valor del atributo *seniority* es incorrecto, saca el siguiente mensaje por pantalla:

      "Incorrect seniority value: <SS>. Valid range is <0>-<MAX_SENIORITY>"

      donde <SS> representa el valor de *seniority* incorrecto, y <MAX_SENIORITY> representa el valor máximo, obtenido de la correspondiente constante de la clase.

   d. Si el valor del atributo *country* es incorrecto, saca el siguiente mensaje por pantalla:

      "Incorrect country value: <CO>. Valid name must contain up <MAX_COUNTRY> letters"

   e. Y *creditCard* de forma similar a *id*.

   Si todos los datos son correctos crea el objeto y preséntalo en pantalla usando el método `toString()`, como en el punto 4 anterior. Prueba adecuadamente el código, modificando manualmente los valores de los datos.

4 Actividad 4: Crear las clases `Employee` y Executive

Tomando como modelo todo lo desarrollado para la clase `Particular`, crea ahora dos clases nuevas para modelar las cotrabajadores empleado y ejecutivo:

- Clase `Employee`, con los siguientes atributos básicos:
  - **Id**, **name**, **seniority**, **country** (como en el caso de `Particular`).
  - **company:** una cadena de máximo 30 caracteres.
  - **bankAccount**: una cadena según el formato de un IBAN, o sea: LLDDDDDDDDDDDDDDDDDDDDD, es decir 2 letras y 22 dígitos (sin guiones de separación).
- Clase `Executive`: los mismos atributos que `Employee`, y además:
  - **passId** : una cadena de 6 dígitos.

5 Actividad 5: Trabajar con arrays de coworkers

Añade código en la clase P2 que haga lo siguiente:

1. Crea 3 arrays estáticos: uno de objetos `Particular`, otro de objetos `Employee` y otro de objetos `Executive`. Todos de tamaño 25. Llama a estos arrays *particulars*, *employees* y *executives*, respectivamente. Decláralos como variables estáticas en la zona de datos globales de la clase P2, de forma que sean visibles para todos los métodos de esta clase.

2. Crea un nuevo método (‘*tipo*’ será un modificador de tipo):

   ```
   public ‘tipo’ void readCoworkersFile (String filename) {...}
   ```

   que lea los cotrabajadores del fichero *coworkers.txt*[1], pasándole en el argumento el nombre de este fichero. Si una línea es un cotrabajador particular (tipo=P), crea un objeto `Particular` y añádelo al array *particulars*; si es un empleado (tipo=E), crea un objeto `Employee` y añádelo al array *employees*; y si es un ejecutivo (tipo=X), crea un objeto `Executive` y añadelo al array *executives*. Crea los objetos y añádelos a los arrays siempre y cuando todos sus datos sean correctos (crea un método `checkParticular(datos)` que compruebe todos los datos y devuelva `true/false`, y lo mismo para las otras clases). Si alguno de los datos de cualquiera de los cotrabajadores es incorrecto[2], simplemente descarta esa línea (cotrabajador) del fichero.

3. Invoca al método `readCoworkersFile` desde el método `main`. ¿Cuál debe ser el modificador de tipo de `readCoworkersFile`? ¿Y el de `checkParticular`?

4. Calcula la media de antigüedad en el centro de coworking de todos los trabajadores, y muéstrala por pantalla según el formato *“`Global average seniority` = resultado”*. Para ello, crea un nuevo método en la clase P2, que calcule y devuelva el valor medio:

   ```
   public float computeAverageSeniority() {…}
   ```

   Fíjate que `computeAverageSeniority` es un método de instancia[3], pero lo llamamos desde un método estático como `main`. Repasa la teoría para saber cómo invocar a `computeAverageSeniority`.

5. Incrementa la antigüedad de los ejecutivos un 10%. Para ello:
   a. Añade el siguiente método a la clase `Executive` (recibe la antigüedad a añadir[4]):

   ```
   public void addSeniority (float extraSeniority) {...}
   ```

---

[1] Este fichero está en el mismo formato que el fichero *coworkers.txt* de la práctica anterior, sin las fechas al final de cada línea.

[2] Usa los **métodos estáticos** de las clases para determinar si los datos de los cotrabajadores son correctos.

[3] Esto no es lo más apropiado, ya que no hay atributos en la clase P2 con los cuales trabaje `computeAverageSeniority`. Se trata simplemente de forzar situaciones para que comprendas las distintas posibilidades.

[4] Si la nueva antigüedad es mayor de 30, debes truncar a 30 (ver clase `java.lang.Math`).

b. Añade el siguiente método a la clase P2 para incrementar la antigüedad en el porcentaje `incr` (entre 0 y 100) de todos los ejecutivos:

```
public static void increaseExecutivesSeniority (float incr) {...}
```

Para cada ejecutivo, debes averiguar su antigüedad, calcular el incremento, y añadirlo llamando a `addSeniority`:

c. Invoca `increaseExecutivesSeniority` en el `main` para proceder al incremento.

6. Guarda en el fichero *executives_output.txt* los cotrabajadores ejecutivos, manteniendo el formato del fichero de coworkers. Guárdalos en el mismo orden en que se leyeron. Si el fichero ya existe, sobreescríbelo. Para ello, añade el siguiente método a la clase P2 e invócalo desde el `main`:

```
public static void writeExecutivesFile (String fileName) {...}
```