Problema 1 (6 puntos)

Apartado a: la clase `Account` (2 puntos)

Diseña e implementa la siguiente clase para representar cuentas corrientes (solamente tendremos en cuenta su identificador y su saldo). Esta clase tendrá los métodos públicos:

- `new Account(String idAccount, int initialBalance)`
  - **tenemos garantizado que `initialBalance` no es negativo e `idAccount` no es null**
  - crea una cuenta con ese identificador y saldo inicial
- `String getId()`
  - devuelve el identificador de la cuenta
- `int getBalance()`
  - devuelve el saldo de la cuenta
- `void deposit(int amount)`
  - **tenemos garantizado que `amount` es positivo**
  - deposita la cantidad dada en la cuenta
- `boolean withdrawal(int amount)`
  - **tenemos garantizado que `amount` es positivo**
  - si no hay saldo suficiente, no hace nada y devuelve falso
  - si lo hay, retira la cantidad dada de la cuenta y devuelve cierto

Apartado b: la clase `AccountsDB` (4 puntos)

Se desea tener una colección ampliable de cuentas para manejar las operaciones que hagamos entre ellas. Fijaos en que estas operaciones deben garantizar las condiciones que deben garantizarse en los métodos de la clase `Account`.

- `new AccountsDB(int initialSize, int sizeIncrement)`
  - **tenemos garantizado que ambos enteros son positivos**
  - crea espacio para guardar inicialmente `initialSize` cuentas
  - en caso de que se necesite más espacio, se incrementará cada vez el tamaño en `sizeIncrement`
- `boolean addAccount(String id, int initialBalance)`
  - si `id` es `null` o existe una cuenta con ese `id` o el saldo inicial es negativo, devuelve falso y no hace nada más
  - si no, añade la cuenta a la base de datos y devuelve cierto
- `boolean transfer(String idFrom, String idTo, int amount)`
  - si alguna de las identificadores `null`, o alguna de las cuentas no existe, o la cantidad `amount` no es positiva, o no hay saldo suficiente en la cuenta `idFrom`, devuelve falso y no hace nada
  - si todo es correcto, transfiere la cantidad de la cuenta `idFrom` a la cuenta `idTo` (por tanto los saldos de ambas cuentas se modifican) y devuelve cierto
- `Account getAccount(String id)`
  - devuelve la cuenta dada correspondiente a ese `id` o `null` si ésta no existe

Usad funciones auxiliares (privadas) para no duplicar código.

Problema 2 (4 puntos)

El programa principal que debéis diseñar tendrá la siguiente estructura:

```
public class AccountsProgram extends CommandLineProgram {
    private AccountsDB accountsDB;
    private int numErrors;

    public void run() {
        inititializeResults();
        ???
    }
    private void initializeResults() {
        accountsDB = loadAcccounts();
        numErrors = 0;
    }
    ???
}
```

En la que la función `loadAcccounts` cargará “mágicamente” las cuentas existentes en una instancia de `AccountsDB` (de esa manera el programa que os pedimos **no ha de leerlas e inicializarlas**).

El programa que debéis completar, ha de:

- pedir al usuario una línea con las transferencias a realizar separadas por espacios
  - cada transferencia tendrá la estructura: `idFrom`**`#`**`idTo`**`#`**`amount`
    - `idFrom:` identificador de la cuenta origen (`String`)
    - `idTo:` identificador de la cuenta destino (`String`)
    - `amount:` cantidad a transferir (`int`)
  - **tenéis garantizado que las tripletas están bien formadas y que tienen los tipos adecuados (aunque podría pasar que no existieran cuentas con alguno de esos identificadores o que las cantidades sean negativas) y ninguno de los identificadores de cuenta incluyen el símbolo #.**
- pedirá al usuario un identificador de cuenta de la que mostrar el saldo

El sistema realizará las transferencias, y mostrará el saldo final de la cuenta dada (o que ésta no existe) y, en caso de haber, el número de transferencias que, por una u otra razón, no han podido realizarse (`numErrors`).

Usad funciones auxiliares en las que descomponer vuestra solución.

Ejemplo de uso, suponiendo unos saldos iniciales de `CC1=500, CA4=300, CI21=0`

```
Transferencias: CC1#CA4#100 CI21#CC1#200 CA4#CI21#350
Cuenta: CI21
El saldo de CI21 es 350 y ha habido 1 error
```

- class **CommandLineProgram**
  - String **readLine**(String message)
  - void **println**(String str)/ void **println**(int n) / void **println**(double d)
  - void **print**(String str) / void **print**(int n) / void **print**(double d)
- class **Double**
  - static double **parseDouble**(String str)
- class **Integer**:
  - static int **parseInt**(String str)
- class **String**:
  - char **charAt**(int index)
  - int **length**()
  - boolean **equals**(String other) / boolean **equalsIgnoreCase**(String other)
  - int **compareTo**(String other)
  - int **indexOf**(char c) / int **indexOf**(String s)
  - String **substring**(int p1, int p2) / String **substring**(int p1)
  - String **concat**(String s) / o usar **+** para concatenar Strings
  - String **trim**()
  - static String **valueOf**(int n)
  - static String **valueOf**(double d)
- class **StringTokenizer**:
  - new **StringTokenizer**(String str)
  - new **StringTokenizer**(String str, String delims)
  - new **StringTokenizer**(String str, String delims, boolean returnDelims)
  - boolean **hasMoreTokens**()
  - String **nextToken**()
- class **Math**
  - static int **max**(int n1, int n2) / static int **min**(int n1, int n2)
  - static double **max**(double d1, double d2)
  - static double **min**(double d1, double d2)