Pràctica 1. `HexDump`: bolcat hexadecimal. Un exemple de streams i buffers binaris
Laboratori d'Aplicacions i Serveis Telemàtics

Josep Cotrina, Marcel Fernàndez, Jordi Forga, Juan Luis Gorricho, Francesc Oller

Introducció

Es vol dissenyar una utilitat `HexDump` en `Java`—fa un bolcat hexadecimal del contingut en bytes d'un arxiu—, que emuli la clàssica utilitat UNIX `hexdump -C arxiu`. Aquesta utilitat es fa servir per explorar el format binari—o textual— dels arxius. Alguns exemples:

```
hexdump -C ~/.wine/drive_c/windows/win.ini

00000000  5b 69 6e 74 6c 5d 0d 0a  73 4c 61 6e 67 75 61 67  |[intl]..sLanguag|
00000010  65 3d 43 41 54 0d 0a 73  43 6f 75 6e 74 72 79 3d  |e=CAT..sCountry=|
00000020  53 70 61 69 6e 0d 0a 73  4c 69 73 74 3d 3b 0d 0a  |Spain..sList=;..|
00000030  73 43 75 72 72 65 6e 63  79 3d 80 0d 0a 69 43 75  |sCurrency=...iCu|
00000040  72 72 65 6e 63 79 3d 33  0d 0a 69 4e 65 67 43 75  |rrency=3..iNegCu|
00000050  72 72 3d 38 0d 0a 69 43  75 72 72 44 69 67 69 74  |rr=8..iCurrDigit|
...
```

Podem observar que la primera columna és l'offset del primer byte de la fila, en hexadecimal. Cada byte esta representat per dos dígits hexadecimals. El tamany d'una fila el considerarem fixe a 16 bytes. A la columna de la dreta hi ha una visualització dels bytes com a caràcters. Aquells que són imprimibles es representen per el seu caràcter mentre que aquells que no ho són es representen per un punt. Podem deduir que es tracta d'un arxiu text en format WINDOWS, ja que el final de línia acaba amb `CR LF`—bytes `0d 0a` (hexa) o `13 10` (decimal)—.

```
hexdump -C /home/pract/LabAST/lastusr98/Desktop/Logout.desktop

00000000  5b 44 65 73 6b 74 6f 70  20 45 6e 74 72 79 5d 0a  |[Desktop Entry].|
00000010  45 6e 63 6f 64 69 6e 67  3d 55 54 46 2d 38 0a 56  |Encoding=UTF-8.V|
00000020  65 72 73 69 6f 6e 3d 31  2e 30 0a 54 79 70 65 3d  |ersion=1.0.Type=|
00000030  41 70 70 6c 69 63 61 74  69 6f 6e 0a 54 65 72 6d  |Application.Term|
00000040  69 6e 61 6c 3d 66 61 6c  73 65 0a 4e 61 6d 65 5b  |inal=false.Name[|
00000050  43 5d 3d 4c 6f 67 6f 75  74 0a 45 78 65 63 3d 2f  |C]=Logout.Exec=/|
...
```

En aquest segon exemple es tracta d'un arxiu de text en format UNIX, ja que el final de línia acaba amb `LF`.

```
hexdump -C "/opt/songs/Ben E. King - Stand by me.mp3"

00000000  49 44 33 04 00 00 00 00  1d 3b 54 49 54 32 00 00  |ID3......;TIT2..|
00000010  00 1a 00 00 03 42 65 6e  20 45 2e 20 4b 69 6e 67  |.....Ben E. King|
00000020  20 2d 20 53 74 61 6e 64  20 62 79 20 6d 65 54 50  | - Stand by meTP|
```

```
00000030  45 31 00 00 00 10 00 00  03 56 61 72 69 6f 75 73  |E1.......Various|
00000040  20 41 72 74 69 73 74 73  54 44 52 43 00 00 00 05  | ArtistsTDRC....|
00000050  00 00 03 32 30 30 30 54  41 4c 42 00 00 00 13 00  |...2000TALB.....|
...
```

En aquest últim exemple es veu clarament que es tracta d'un arxiu binari de música—en aquest cas en format MP3—. Fixem-nos en la secció de *metadades* ID3 i els `tags` TIT2—titol—, TPE1—autor— o TDRC—any d'enrregistrament—.

La nostra aplicació JAVA s'invocarà amb la línia p.e. `java HexDump "/opt/songs/Ben E. King - Stand by me.mp3"`—o bé afegint l'argument d'arxiu en els entorns NETBEANS o ECLIPSE—.

L'objectiu principal de la pràctica és el de programar una cua *FIFO*—First In First Out— de bytes i usar els seus mètodes principals. La programació d'aquesta cua és bàsica per implementar els protocols de transport que veurem més endavant.

Idees d'implementació

A continuació s'exposen algunes idees d'implementació que poden ser útils per a la realització de la pràctica però que en cap cas són obligatòries. Si ajuden, feu-les servir. Si per el contrari confonen baseu-vos en el vostre propi criteri. L'únic requeriment obligatori és programar una classe `ByteQueue` i usar els seus dos mètodes principals: `public int get(byte[] b, int off, int len)` i `public void put(byte[] b, int off, int len)` com veurem més endavant.

La implementació més simple (recomanada)

La idea seria dimensionar la cua de bytes `ByteQueue` a 16 i programar dues activitats en un bucle:

```
...
// initialize file of bytes
InputStream in = ...;

// create byte queue
ByteQueue bq = ...;

while (input(in, bq))
   output(bq);
...
```

input(InputStream in, ByteQueue bq): llegeix del stream `in` un array de bytes que diposita a la cua `bq`. Retorna una indicació de fi si no hi han més bytes a llegir.

output(ByteQueue bq): llegeix un array de bytes de la cua `bq` i escriu per consola una línia segons el format de `hexdump -C`.

Una implementació més complexe (opcional)

Volem independitzar el tamany del buffer de lectura del fitxer o escriptura a la consola del tamany de la cua de bytes. En general pensem en activitats que es comuniquen a través de buffers de bytes. Un d'ells és la cua `bq`. N'hi han d'altres que no tenen perque ser de tipus `ByteQueue` que serveixen per comunicar activitats.

El buffer de lectura del fitxer s'haurà de dipositar a la cua `bq` per fragments. Tanmateix s'hauran d'extreure fragments de la cua `bq` per passar-les al formatejador de consola. El bucle principal d'aquest possible disseny seria:

```
...
InputStream in = ...;
ByteQueue bq = ...;
// altres buffers
```

```
...

while (read(in, /* buffer lectura */)) {
   produce(/* buffer lectura */, bq);  // diposita un fragment a bq
   consume(bq, /* buffer escriptura */); // extreu un fragment de bq
   format(/* buffer escriptura */);  // formateja a consola
}
...
```

A continuació es dona feta una classe `SimpleByteQueue` que posa i treu bytes un-a-un i pot servir de base per a la programació de `ByteQueue`:

```
package ast.util;

public class SimpleByteQueue implements Queue<Byte> {

   byte[] buffer;
   int rpos, bcount;

   public SimpleByteQueue() {
      this(1024);
   }

   public SimpleByteQueue(int capacity) {
      buffer = new byte[capacity];
      // default values: rpos = 0 ;
      //             bcount = 0 ;
   }

   public Byte get() {
      if (bcount == 0) throw new IllegalStateException("buffer empty");
      byte b = buffer[rpos];
      rpos = (rpos + 1) % buffer.length;
      bcount --;
      return b;
   }

   public void put(Byte oneByte) {
      // check buffer has enough free space (buffer.length - bcount >= 1)
      if (buffer.length == bcount) throw new IllegalStateException("buffer full");

      buffer[(rpos + bcount) % buffer.length] = oneByte;
      bcount ++;
   }

   public int size() {
      return bcount;
   }

   public int free() {
      return buffer.length - bcount;
   }

   public boolean empty() {
      return bcount == 0;
```

```
    }

    public boolean full() {
        return bcount == buffer.length;
    }

    public Byte peekFirst() {
        if (bcount > 0) {
            return buffer[rpos];
        } else {
            return null;
        }
    }

    public Byte peekLast() {
        if (bcount > 0) {
            int wpos = (rpos + bcount - 1) % buffer.length;
            return buffer[wpos];
        } else {
            return null;
        }
    }
}
```

L'interfície de la qual hereten `SimpleByteQueue` i `ByteQueue` és:

```
package ast.util;

public interface Queue<T>
{

    /**
     * Quantitat d'elements en la cua
     */
    public int size();

    /**
     * Està totalment buida ?
     */
    public boolean empty();

    /**
     * Està totalment plena ?
     */
    public boolean full();

    /**
     * Obté primer missatge de la cua (ó null si la cua és buida)
     */
    public T peekFirst();

    /**
     * Obté últim missatge de la cua (ó null si la cua és buida)
     */
    public T peekLast();
```

```
    /**
     * Obté i elimina primer missatge de la cua
     * Llença IllegalStateException si la cua és buida
     */
    public T get();

    /**
     * Introdueix <value> al final de la cua
     * Llença IllegalStateException si la cua és plena
     */
    public void put(T value);

}
```

Es demana:

1 ByteQueue

Programeu la classe `ByteQueue` amb dos mètodes addicionals `int get(byte[] b, int off, int len)` i `void put(byte[] b, int off, int len)` seguint la següent plantilla:

```
package ast.util;

public class ByteQueue implements Queue<Byte> {

    ...

    /**
     * Obté un màxim de len bytes a partir de la posició
     * off de l'array b. Retorna el nombre de bytes llegits
     * i eliminats de ByteQueue
     * Llença IllegalStateException si la cua és buida
     */
    public int get(byte[] b, int off, int len) { ... }

    /**
     * Afegeix len bytes a partir de la posició
     * off de l'array b al final de la cua
     * Llença IllegalStateException si la cua és plena
     */
    public void put(byte[] b, int off, int len) { ... }

    ...
}
```

No hereteu de `SimpleByteQueue`. Afegiu a `ByteQueue` el codi de `SimpleByteQueue` que considereu útil.

2 HexDump

Programeu l'aplicació de bolcat hexadecimal fent servir `ByteQueue` i els mètodes `int get(byte[] b, int off, int len)` i `void put(byte[] b, int off, int len)`.

Entorns de programació

Les aplicacions de consola es poden executar en tres entorns diferents, a triar segons les indicacions del professor:

NETBEANS: la consola de NETBEANS és especial i no respon a `Cntl-D`.

ECLIPSE: la consola de ECLIPSE si respon a `Cntl-D`.

consola: es pot programar amb NETBEANS/ECLIPSE i executar sobre la consola. S'ha de tenir en compte la distribució d'arxius de NETBEANS/ECLIPSE:

NETBEANS:

```
NetBeansProjects/
└── Streams
    ├── build
    │   └── classes
    │       ├── Streams_1.class
    │       ├── Streams_2.class
    │       ├── Streams_3.class
    │       ├── Streams_4.class
    │       ├── Streams_5.class
    │       └── Streams_6.class
    ├── dist
    │   └── ...
    ├── nbproject
    │   └── ...
    ├── src
    │   ├── Streams_1.java
    │   ├── Streams_2.java
    │   ├── Streams_3.java
    │   ├── Streams_4.java
    │   ├── Streams_5.java
    │   └── Streams_6.java
    └── ...
```

Ens hauriem de situar en el directori del projecte: `.../NetBeansProjects/Streams$` i executar: `.../NetBeansProjects/Streams$ java -cp build/classes Streams_1` p.e.

ECLIPSE:

```
workspace/
└── Streams
    ├── bin
    │   ├── Streams_1.class
    │   ├── Streams_2.class
    │   ├── Streams_3.class
    │   ├── Streams_4.class
    │   ├── Streams_5.class
    │   └── Streams_6.class
    ├── src
    │   ├── Streams_1.java
    │   ├── Streams_2.java
    │   ├── Streams_3.java
```

```
│   ├── Streams_4.java
│   ├── Streams_5.java
│   └── Streams_6.java
└── ...
```

Ens hauriem de situar en el directori del projecte: `.../workspace/Streams$` i executar: `.../workspace/Streams$ java -cp bin Streams_6` p.e.