Ejercicios de introducción

Programación en Java para la asignatura Algoritmos y estructuras de datos (AyEdD)

Gorka Guardiola Múzquiz

1 de Febrero de 2022

Este documento contiene el tercer ejercicio obligatorio de la asignatura.

Este ejercicio es para que comiences a mostrar cierto conocimiento de java (tipos de datos básicos).

Ejercicio 3

Base32 Escribe una clase en Java llamada `Base16.java`. El objetivo de estar clase es codificar y decodificar bytes en una codificación en Base16. Esta codificación es similar a Base64 (ver `https://es.wikipedia.org/wiki/Base64`). Aunque este tipo de codificaciones, formalmente son equivalentes a un cambio de base (de ahí el nombre), queremos que pienses en ello como otra forma equivalente de codificar la misma información que hay en los bits originales con otros bits (en este caso utilizando caracteres).

Los bytes de entrada se considerarán juntos como un conjunto de bits. Se dividirán en grupos de 4 bits (añadiendo algunos bits a cero al final en caso de ser necesario) y se codificarán como caracteres siguiento la siguiente tabla

0000	0001	0010	...	0101	0110	0111	...	1110	1111
'0'	'1'	'2'	...	'5'	'A'	'B'	...	'I'	'J'

Para la implementación **no puedes utilizar una tabla gigante**, ya sea en forma de `switch`, `if` gigante o algo parecido. Recuerda que los caracteres están ordenados y se pueden comparar, por ejemplo:

```
public static final int FirstChar = 'l';
public static final int LastChar = 'z';
...
if(ch >= FirstChar && ch <= LastChar){
...
}
```

Utilizar una tabla gigante y/o no utilizar constantes hace el código más difícil de modificar y fácil cometer errores. Piensa en qué sucede si te piden que cambies el rango de caracteres.

La clase tiene que tener implementados los siguientes métodos:

```
public static String encode(byte[] content)   Codifica los bytes en base16
public static byte[] decode(String text)      Decodifica los bytes en base 16
```

En caso de que el argumento sea erróneo, devolverán `null`.

La clase que entregues puede contener `main` o no, (lo puedes utilizar para probar tu código). El test y la corrección lo ignorará.

Por ejemplo, si codifico el array:

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
```

Obtengo la string:

```
"0001020304050A0B0C0D0E0F0G0H0I0J10111213"
```

y viceversa.

Pautas de implementación: Para codificar en binario, te recomendamos que utilices la operación de shift (por ejemplo $<<$ y $>>$) y los *and* y *or* bit a bit (los operandos binarios infijos & y |). Recuerda que si sabes que el contenido tiene sentido, puedes utilizar *cast* entre tipos básicos; al final todo son bits. Ejemplos:

```
byte b;
int i;
i = 2452345;
b = (byte)(i & 0xff); //me quedo con el byte menos significativo del entero
b = (byte) (0x0f)(i >> 4) //me quedo con los bits 4 5 6 y 7
```

Ojo, no hagas que tu código tenga una tabla gigante. Recuerda que se pueden comparar los caracteres como se ve arriba y hacer álgebra con ellos (si hace falta con un *cast* de por medio). Por ejemplo:

```
byte b = 3;
char c = (char)('A' + b);
StdOut.println(c);
```

Entrega: Recuerda que hay que entregar mediante el sistema de entregas y en el aula virtual. Si no está en ambos, no se considera entregado. Recuerda ejecutar `rprueba.sh` periódicamente y para realizar la entrega final y `prueba.sh` para probar tu práctica. Recuerda también compilar con el parámetro `-encoding utf-8`.