Utils class

We create a new class, non-instantiating, that contains static methods for different purposes. The methods are:

```
public static void delay(int ms)
     puts the process to sleep for the given milliseconds in ms
     use the method Thread.sleep(ms) which needs to be enclosed in a try-catch construct
```

```
public static boolean isANumber(String s)
     returns true if s contains only digits 0-9; false otherwise
```

```
public static String leftPad(String text, int desiredLength, char paddingItem)
     returns text padded to the left with the paddingItem so that the total length of the
     padded string becomes desiredLength
     if text.length() > desiredLength then text is returned unchanged
     if text.length() == 0 then the returned string is as many paddingItems as specified
     in desiredLength
```

```
public static String removeChars(String text, char c)
     returns text with all occurrences of c removed
```

(more methods next page)

```
public static String longToBytesStr(long num)
     returns the eight byte ASCII representation of num; this is done by taking every
     byte of num individually and creating an eight byte string. For example:
```

num	**longToBytesStr()**	**explanation**
4846247313457894228	CATALYST	4846247313457894228 in hexadecimal is 0x434154414C595354 starting from the left: 0x43 is the ASCII code for 'C' 0x41 is the ASCII code for 'A' 0x54 is the ASCII code for 'T' 0x41 is the ASCII code for 'A' 0x4C is the ASCII code for 'L' 0x59 is the ASCII code for 'Y' 0x53 is the ASCII code for 'S' 0x54 is the ASCII code for 'T'
5359685314039263287	James007	5359685314039263287 in hexadecimal is 0x4A616D6573303037 starting from the left: 0x4A is the ASCII code for 'J' 0x61 is the ASCII code for 'a' 0x6D is the ASCII code for 'm' 0x65 is the ASCII code for 'e' 0x73 is the ASCII code for 's' 0x30 is the ASCII code for '0' 0x30 is the ASCII code for '0' 0x37 is the ASCII code for '7'

Most of the string representations will be unreadable characters. Use the table above for testing. The pseudo-code is on the next page:

```
s = null string

let byte b = isolate the most significant byte by performing a bit-wise AND (single &) of the given
number with 0xFF00000000000000L (conversion to hexadecimal of the given number is not necessary). For
example: byte b = number & 0xFF00000000000000L
let b = shift b to the right by 56 bit positions: b = b >> 56
s = s + (char) b

let b = isolate the next most significant byte by performing a bit-wise AND with 0xFF000000000000L
let b = shift b to the right by 48 bit positions: b = b >> 48
s = s + (char) b

let b = isolate the next most significant byte by performing a bit-wise AND with 0xFF0000000000L
let b = shift b to the right by 40 bit positions
s = s + (char) b

let b = isolate the next most significant byte by performing a bit-wise AND with 0xFF00000000L
let b = shift b to the right by 32 bit positions
s = s + (char) b

let b = isolate the next most significant byte by performing a bit-wise AND with 0xFF000000L
let b = shift b to the right by 24 bit positions
s = s + (char) b

let b = isolate the next most significant byte by performing a bit-wise AND with 0xFF0000L
let b = shift b to the right by 16 bit positions
s = s + (char) b

let b = isolate the next most significant byte by performing a bit-wise AND with 0xFF00L
let b = shift b to the right by 8 bit positions
s = s + (char) b

let b = isolate the least significant byte by performing a bit-wise AND with 0xFFL
no shifting needed
s = s + (char) b

return s


(Notes:  The L at the end of the hexadecimal numbers denote a long constant. This is needed in Java
         so that all 64 bits will be taken into account
         This method can be written with only one return statement)
```