

Funciones de las consultas con el tipo:

```
private static ArrayList Funcion {  
    -----  
    for ( ... )  
        ....add()  
}
```

* No utilizar get veces en las consultas. Preguntarle al monstruo.

• En la 4ª consulta:

Lo podemos hacer montando un submenú con la siguiente estructura:

1: botas 2: casco 3: manos ...

case 1:

case 2:

:

• Variables que tenemos en el MENU de main:

- Scanner
- opcion
- entrada

3ª Sesión

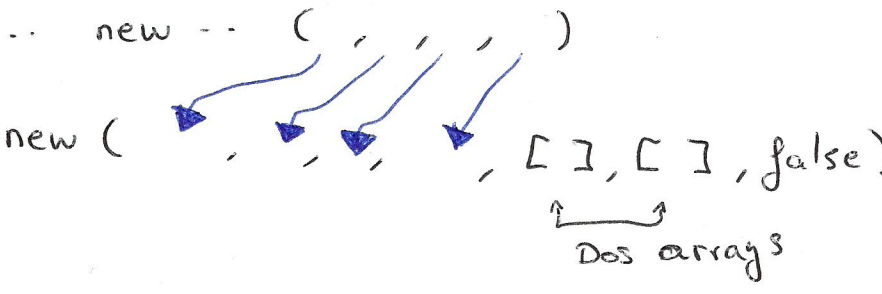
[En Ruby el último método con el mismo nombre que otro,
anula al otro.]

• Para inicializar en vez de sobrecargar lo hacemos con 3 métodos asociados a la clase (no existen métodos estáticos).

(Constructores del punto 2 de la 3ª sesión):

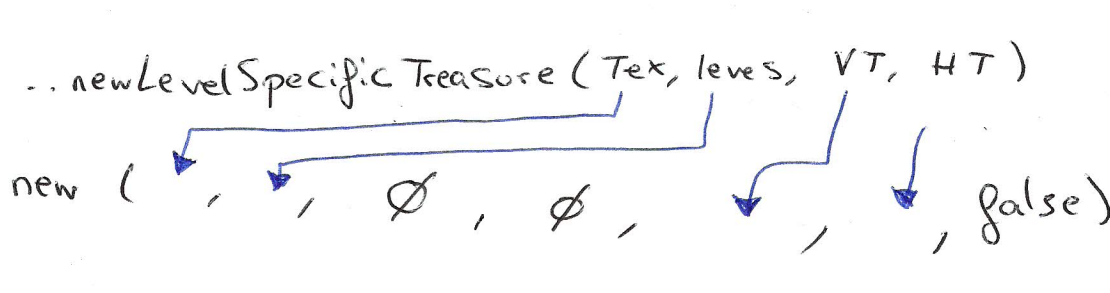
①

```
def BC ... new ... (
  new ( , , , , [ ], [ ], false )
end
```



②

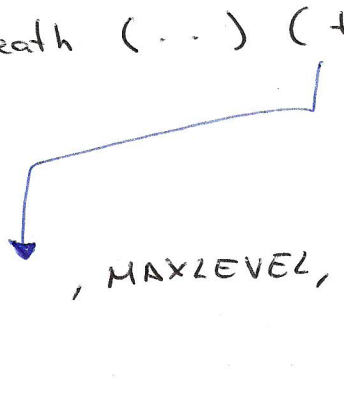
```
def BC .. newLevelSpecificTreasure (Tex, leves, VT, HT)
  new ( , , , , , , false )
end
```



③

```
def BC Death ( ... ) ( + )
  new ( , MAXLEVEL, MAXERASE, ..., [ ], [ ], true )
end
```

↳ nos máximos o declaramos nosotras
variables estáticas: ~~maxLevel~~
maxLevel



(utilizar attr_reader en todo lo que se necesite):

MAIN RUBY

PruebaNapakalaki

Examen en
Ubuntu 16

• plugin Ruby en el
aula están en
lsi/pdoo

• venir a las 15:15 para
ir preparando.

usar (require Prize, Monster ...)

Class PruebaNapakalaki

@@ Monster (: []
@array.new)

Podemos usar las 2 opciones
pero mejor la 2ª.

Ahora tendremos 4 funciones con la misma estructura:
para las consultas.

def self. [] ()

(PN) → se podría poner también PruebaNapakalaki.

Main:

def self main()

[Declaración de monstruos
dentro del array monster] Esto se puede hacer aquí
o fuera, como queramos.

[MENÚ] Para leer desde teclado se usa chomp.

end

end

Para que todo esto funcione hay que ejecutarlo poner el
final fuera de la clase: PruebaNapakalaki.main

(Si no lo ponemos es como si no hubiésemos escrito nada.)

(Si Tenemos los monstruos fuera del main: PruebaNapakalaki.monstruos
PruebaNapakalaki.main