EXAMEN PDOO : TEORÍA 1 ~ [2019 - 2020] (SOLUCIONES)

TIPO DE EXAMEN: 1

A continuación, se muestran las soluciones al examen de `TEORÍA1` de `PDOO`:

Pregunta 1 - [7 puntos] - Apartado 1

*SOLUCIÓN: Se nos pide crear otro constructor para la clase `Examen1` que reciba 3 parámetros que son **(a,b,filtro)** Por lo tanto, la implementación sería la siguiente:*

```
def self.constructor(a,b,filtro)
     #PRIMER PASO: FILTRADO
     a_filtrado = filtra_a(a,filtro)
     #LLAMAMOS AL OTRO CONSTRUCTOR
     new(a_filtrado,b)
end
```

Pregunta 1 - [7 puntos] - Apartado 2

SOLUCIÓN: En este apartado sólo se nos pide definir el método, dejando la implementación vacía puesto que desconocemos el propósito de dicha función. La resolución sería la siguiente:

```
def self.filtra_a(a,filtro)
   #IMPLEMENTATION GOES HERE
end

#ADEMÁS, PODRÍAMOS PONER ESTE MÉTODO PRIVADO DE CLASE, PUESTO QUE SÓLO LO
UTILIZARÍAMOS PARA EL CONSTRUCTOR
method_class_private :filtra_a
```

PREGUNTA 2 - [7 puntos] - Apartado 1

SOLUCIÓN: Tenemos que ver la salida de la sentencia:

```
puts Examen1_Hija.new(77).salida
```

Se ha redifinido el initialize (con un argumento) de la clase Hija. Clase para la cual se define un atributo de instancia llamado "c".

Por tanto, lo que haría Examen1_Hija.new(77), sería inicializar el atributo de @c a un valor "valor_c" que es obtenido del método ajuste_del_atributo_c(77). Ahora, la clase Examen1_Hija hereda los métodos del padre, pero no sus atributos puesto que son de instancia, y en el constructor de la clase hija no se ha realizado ninguna llamada a super.

Es por lo tanto que esa sentencia produciría un error al ejecutarla.

PREGUNTA 3 - [3 puntos] - Apartado 1

*SOLUCIÓN: Utilizando la herencia de Ruby (sabiendo que `Examen1_Hija` hereda de `Examen1`, la implementación del método `salida(a)` de la clase hija se puede hacer muy sencilla utilizando la llamada a **super**. Por lo tanto:*

```
#METODO SALIDA DE EXAMEN1_HIJA
def salida(a)
    return super(a) - 1
end
```

PREGUNTA 3 - [3 puntos] - Apartado 2

SOLUCIÓN: No se puede hacer en ninguna de las dos clases , ya que en ruby no existe el concepto de sobrecarga (en java sí se podría).

PREGUNTA 4 - [3 puntos] - Apartado 1

En este apartado como en el otro se nos pregunta si hay errores en la sentencia:

```
ex4.getContenedor().add(44);
```

*SOLUCIÓN: Tenemos que tener en cuenta que `Examen4` como `Examen3` están en el mismo paquete `Examen`. En este caso en concreto el atributo **contenedor** tiene visibilidad de **PAQUETE**, por lo que no se produciría ningún error de ningún tipo.*

PREGUNTA 4 - [3 puntos] - Apartado 2

*SOLUCIÓN: La misma situación que la descrita anteriormente sólo que el atributo **contenedor** ahora tiene visibilidad **PRIVADA**. La tentación natural es pensar que se produciría un error puesto que este atributo ahroa pasa a ser privado y por tanto no se puede añadir el número. SIN EMBARGO, tanto en Java como en Ruby, **TODO SON REFERENCIAS**, y cuando se obtiene una referencia a un atributo (con el método `getContenedor()` en este caso), las restricciones impuestas ya no tienen efecto. En conclusión, se ejecutaría correctamente como en el caso anterior.*