EXERCICI PUNTUABLE TECPRO

23/04/2018 Grau en Enginyeria de Sistemes TIC

COGNOMS:

NOM: GRUP de LAB:

Exercici 1. El diagrama UML. Una empresa que es dedica a la venda i reparació de sistemes de seguretat us ha encarregat una aplicació que gestioni els encàrrecs que rep dels seus clients. L'aplicació ha d'emmagatzemar informació dels sistemes de seguretat que té l'empresa, dels seus clients i dels serveis que aquests requereixen.

Un sistema de seguretat té un codi alfanumèric que l'identifica i està format per un conjunt de sensors dels quals n'hi ha de tres tipus diferents: sensors volumètrics, que s'utilitzen per detectar moviment, sensors d'infraroig, que detecten calor i sensors tàctils, que es disparen si dues superfícies deixen d'estar en contacte (per exemple quan s'obre una porta).

Tots aquests sensors tenen un preu i un codi alfanumèric que els identifica. Els sensors volumètrics es poden configurar amb un valor real que indica la longitud en metres del seu radi d'acció. Els d'infraroigs es poden configurar amb un valor enter de manera que l'alarma es dispara quan la temperatura del lloc on es troba el sensor supera aquest valor. Els sensors tàctils, tenen un LED ocult al client que es posa de color verd si l'alarma va saltar per un canvi de temperatura, o bé vermell per la detecció de moviment. D'aquesta manera és fàcil saber la causa de l'alarma.

Quan un client nou es posa en contacte amb l'empresa, cal emmagatzemar el seu nom, els cognoms, el DNI, l'adreça, la població i el seu telèfon. Els clients poden demanar dos tipus de servei a l'empresa: la compra d'un sistema de seguretat, o bé la reparació d'un que ja tenien instal·lat. Interessa guardar tots els serveis que cada client sol·licita al llarg del temps, la data en què es va sol·licitar cadascun d'ells, el codi numèric que identifica cada servei i el sistema de seguretat que es va vendre o reparar. Cal notar que un sistema de seguretat només es pot vendre un cop i, en canvi, ser reparat moltes vegades al llarg del temps.

Dibuixeu el diagrama UML que doni resposta a les especificacions anteriors. Per cadascuna de les classes creades, afegiu una breu indicació de quins atributs contindria cada classe, per permetre les relacions que heu detectat.

Exercici 2. Recursivitat. [Apartat a] Dissenya òptimament la funció recursiva *suma*, tal que donat un nombre natural, retorni la suma dels seus dígits. Només es pot utilitzar la funció str/int/len. Qualsevol altra funció predefinida comportarà una puntuació nul·la.

[Apartat b] Dissenya òptimament la funció recursiva *compress*, que permeti comprimir un text, de la següent manera, cada vegada que una lletra es repeteix, es guarda la indicació de quants cops s'ha repetit (si és superior a 1).

```
def suma(x):
    """
    >>> suma(371)
    11
    >>> suma(3)
    3
    """
```

```
def compress(s):
    """
    >>> compress('abc')
    'abc'
    >>> compress('abbcddd')
    'a2bc3d'
    """
```

Exercici 3. Classes, objectes, relacions i excepcions. Donada la següent definició de classes, [Apartat a] Escriu el resultat de les expressions contingudes en el main i que estan numerades. Si no retorna res, escriu None. Si es tracta d'un error, escriu Error i una breu explicació del perquè.

```
class F(object):
    x = 0
    y = 10
    def __init__(self, y):
        self.x = y
    def  a1(self, y):
        self.x = max(self.x, y)
        return self.x
    def a2(self):
        self.x = max(self.x, self.y)
        return self.x
    def invoca(self):
        try:
            return self.espera(self.x)
        except Exception as e:
            return "not found"
    def __str__(self):
        return self.__class__.__name__+"--"+str(self.x)

class G(F):
    def  b(self):
        self.y = self.x * self.x
        return self.y
    def espera(self,x):
        return x+4

class H(F):
    def espera(self,x):
        return x*2

class I(H):
    pass
```

```
if __name__=='__main__':
    #1.1  f=F(5)
    #1.2  g=G()
    #1.3  print f.a1(7)
    #1.4  g=G(3)
    #1.5  print f.a1(3)
    #1.6  print g.a1(5)
    #1.7  print f.b()
    #1.8  print g.b()
    #1.9  print f.a2()
    #1.10 print g.a2()
    #1.11 print g.invoca()
    #1.12 h=H(22)
    #1.13 print h.invoca()
    #1.14 print g
    #1.15 print h
    #1.16 i=I(5)
    #1.17 print i.invoca()
```

[Apartat b] En la sentència *i=I(5)* de la línia 1.16, a quin mètode es crida? Justifica si aquest mètode és un exemple de sobrecàrrega/redefinició/herència/delegació de mètodes.

[Apartat c] En la sentència *print h.invoca()* de la línia 1.17, a quin mètode es crida? Justifica si aquest mètode és un exemple de sobrecàrrega/redefinició/herència/delegació de mètodes.

[Apartat d] Ara afegim la següent definició de classe contenidora elsF, i se us demana justificar què s'escriurà per pantalla en executar el main.

```
class elsF(object):
    def __init__(self):
        self.dades={}

    def add(self,dada):
        if dada not in self.dades:
            self.dades[dada]=repr(dada)
        else:
            raise Exception("already")

    def __iter__(self):
        return iter(self.dades)

    def __str__(self):
        return "Els fs info "+">>>".join([str(element) for element in self])
```

```
if __name__=='__main__':
    k=elsF()
    r=I(1)
    j=F(1)
    try:
        k.add(r)
        k.add(j)
        k.add(j)
        k.add(r)
    except Exception as e:
        print e
    print k
```

[Apartat e] Dibuixeu el diagrama UML resultant que contingui les classes anteriors.