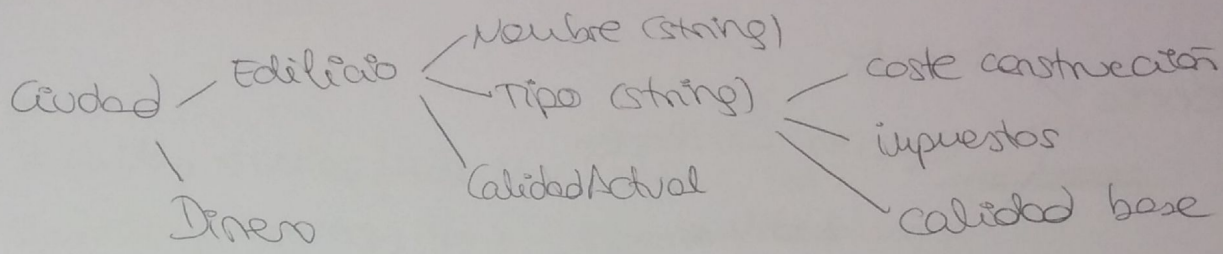
CIUDADMATIC

1/6/15 SCF



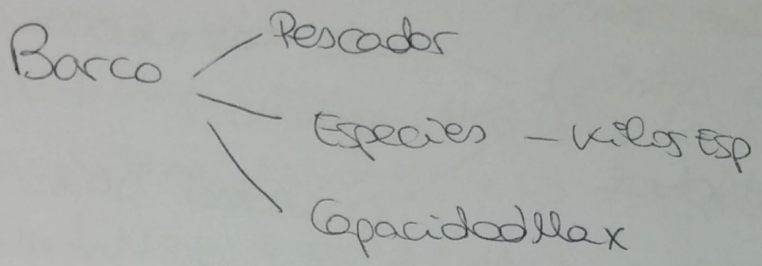
Atributos

```
int _dinero

struct infoTipo {
    int coste;
    int impuestos;
    int calidadBase;
};

HashMap <string, infoTipo> _tipos;

struct infoEdificio {
    string TipoEdificio;
    int calidadAct;
    NodoAB *pAntigüedad;
};

HashMap <string, infoEdificio> _edificio;

Lista <string> _ListaAntigüedad;
```

Si usamos una lista para guardar los edificios por antigüedad, el método 'Fin turno' tiene complejidad cuadrática (para cada edificio, si su calidad llega a 0, hay que buscarlo en la lista y eliminarlo.

Para solucionarlo usamos una clase Nodo interna y listas doblemente enlazadas y en la estructura infoEdificio guardamos un puntero al nodo de la LE en el que está el edificio.

BARCOMAR



especie kgs

```
HashMap < string, HashMap <string, int> > _pescador;
          pescador
```

```
HashMap < string,int > - EspecieKilos;
HashMap <string, int> _PescadorKilos;
int _capacidadMax;
```

BARCOMATIC

```
class BarcoMatic {

  private:
  HashMap <string, HashMap <string, int>> _barco;
  HashMap <string, int> - EspecieKilos;
  HashMap < string, int > - PescadorKilos;
  int pesoMax;

  public:
  barcoMatic();
  void altaPescador(string Pescador);
  void nuevaCaptura(string Pescador, int peso, string especie);
  list <Pareja <string, int>> capturasPescador(string pescador);
  int kilosEspecie (string especie);
  int kilosPescador (string pescador);
  int bodegaRestante ();
};
```

```
BarcoMatic :: BarcoMatic (int n)
{
    peso = n;
}

void BarcoMatic :: altaPescador (string Pescador)
    if (!_barco.contains(pescador)) {
        _barco.insert (pescador, HashMap <string, int>);
        int kilos = 0;
        _kilosPescador.insert (pescador, kilos);
```

```
void BarcoMatic::nuevaCaptura(string pescador, int peso, string especie)
{
    if (!barcos.contains(pescador))
        throw NoEstePescador();
    if (pesoBarco - peso) < 0 {
        throw BodegaLlena();
    }
    int pesoAux;
    HashMap<string, int> aux = _barco.at(pescador);
    if (aux.contains(especie)) {
        pesoAux = aux.at(especie) + peso;
    }
    else {
        pesoAux = peso;
    }
    aux.insert(especie, pesoAux);
    _barcos.insert(pescador, aux);          //Actualizo el HashMap
    _kilosPescador.[pescador] += peso;

    if (_especieKilos.contains(especie)) {
        pesoAux = _especieKilos.at(especie);
        pesoAux += peso;
    }
    else {
        pesoAux = peso;
    }
    _especieKilos.insert(especie, pesoAux);
    pesoMax = pesoMax - pesoAux;
}
```

```
list < Pareja <string, int>> capturasPescador (string
                                                pescador)
{
    if (b_barcos.contains(pescador)) {
    } throw NoEstaPescador;
    list <Pareja <string, int>> listaEspkgs;
    HashMap <string, int>::Iterator it = [illegible].
                                                    it)
    _barco.at(pescador).begin([illegible])

    while (it != _barco.at(pescador).end()[illegible]) {
        Pareja <[illegible], [illegible]> ParejaAux = new Pareja [illegible]
                string int                                    <string, int>
                                                    [illegible]
                                                (it.key(), it.value())
        listaEspkgs.push_front(ParejaAux);
        it.next();
    }

return listaEspkgs;
}
```

```
int kilosEspecie (string especie){
    if (!_barco.contains(especie))
        throw NoEspecie();

    return _EspecieKilos.at();
                         especie
```

```
int kilosPescador (string pescador){
    if(!barco.contains(pescador)){
        throw NoHayPescador;
    }
    return _PescadorKilos.at(pescador);
```

```
int BodegaRestante () {
    return pesoMax; // Ya lo he iba
                    actualizando en
                    nueva captura
```