Índex

Contingut

01. Decàleg del treball en equip

Quan es treballa en equip, i el projecte va creixent de mida i de complexitat, convé tenir una sèrie de nocions i de normes clares. Moltes normes son de pur sentit comú. Altres normes son d'aplicació de marcs de treball o de decisions tècniques. S'han de complir totes.

Si no saps pregunta. Si dubtes pregunta. No programis sense pensar abans. Si trobes que estès repetint la mateixa estructura en llocs diferents, alguna cosa no estàs fent bé. Si trobes que necessites crear classes que no concorda amb l'estructura establerta, alguna cosa no està bé.

Períodicament s'ha de refactoritzar. S'ha de repassar el codi i millorar-lo. Es moment de trobar estructures ineficients o repetides i crear codi de més qualitat. Si no refactoritzes mai, probablement el codi te errades.

Assumeix cicles de desenvolupament curts. Ves de més detall a menys. Centra't en un detall i acaba'l. Després passa al detall que hi ha per sobre i acaba'l. No deixis coses a mitges. Usa TODO's

02. Documenta les decisions

Es molt important deixar constància escrita de les decisions que organitzen i normalitzen el projecte. S'ha de crear un document de "decàleg", on l'equip acorda certes normes bàsiques de tractament del codi, dels arxius que es generen, de l'organització del repositori, etc.

03. Estructura del projecte

Sigui quina sigui l'estructura del projecte finalment triada, un cop triada no es pot canviar i tothom l'ha de seguir.

Proposta d'organització:

- Projecte
  - BBDD: Scripts de la BBDD i generadors de script
  - Controladors: Controladors del model MVC
  - Documentació: Documentació i diagrames
  - Excepcions
  - Interfaces
  - Model: Classes del model MVC
  - Utils: Utilitats que fan ús des de MVC
  - Vista: Vistes del model MVC

04. Decideix els noms i crea document.

Obliga i obliga't a emprar sempre la mateixa nomenclatura

- Nom de carpetes i paquets
- Nom de classes. Sobre tot: Ús de prefixos i idioma
- Normes per a l'ús de Interfícies. Una classe de domini: Que està obligada a implementar?
- Ús de herència: Certes classes hereten obligatòriament d'una altra abstracta que ja implementa funcionalitats comunes
- Nom de mètodes
- Norma de sobrecàrrega de mètodes
- Nomenclatura de paràmetres, (ús de prefixos)
- Nomenclatura de propietats i variables privades de propietat, (ús de prefixos, ús de get/set, etc)

05. La BBDD està sotmesa al control de codi

No programis directament el SGBD
Crea scripts
Automatitza la creació del script general
Manté tots els scripts en el repositori de codi
A partir d'un cert punt, (quan es determini en l'equip, però sobre tot quan l'aplicatiu ja es troba parcialment en producció), està prohibit substituir taules, i has de crear ALTERs. Els alters s'afegeixen al script.de la taula. Cada taula te el seu script
Les dades de migració i de "punt 0" també es mantenen al script
Cada cert temps, si és possible, els ALTERs es consoliden i es torna a crear taula.
Empra procediments emmagatzemats sempre que sigui possible

Estructura de la carpeta BBDD

- 🗁BBDD: Scripts de la BBDD i generadors de script
  - 🗁Creació: Scripts generals per a crear la BD, passar dades inicials, etc.
  - 🗁Procediments: Scripts dels procediments emmagatzemats
  - 🗁Taules: Scripts de definició de taules i alers
    - 🗁Relacions: Scripts de definició de les relacions entre taules
  - 🗁Vistes: Scripts de definició de les vistes

Per a cada entitat del Model hauries de tenir:

- Un script per la definició de la persistència en BBDD, (la taula o taules).
- Un script per la selecció a partir de PK i per a les cerques i filtres, (model_load.sql). Generalment en el mateix script, però poden estar en dos, depenent del que els filtres i cerques necessitin retornar.
- Un script per a l'alta i edició, (model_save.sql). Si és possible, empra cursors per al INSERT/UPDATE, i fes que el PROCEDURE tingui un únic

paràmetre XML. Automatitza el INSERT/UPDATE fent un SELECT previ per comprovar si el registre existeix
- Un script per a l'esborrar, (model_erase.sql)
- Inclou a cada script la conveniència o no de verificar si existeix, i l'acció a fer, (no fer res, esborrar, ...).

05.1. Una proposta per a la BBDD

- Un script per **esborrar** la BBDD

```
drop schema if exists bbdd;
```

- Un script de **creació** de la BBDD

```
CREATE SCHEMA bbdd;
use bbdd;
```

- Un script per a **bolcar les dades generals i paràmetres**

```
INSERT INTO PerfilsUsuari (IDPerfil, Perfil, EsAdministrador) VALUES('10', 'Administradors', 1);
INSERT INTO Usuaris (Login, Clau, Nom, Actiu, CanviContrasenya, IDPerfil, IDPestanyaDefecte, PestanyaCollapsed) VALUES('Admin', 'admin', 'Usuari administrador per defecte', 1, 0, '10', '10', null);
```

Etc

- Un script per la **definició de les taules base** i un altre script per la **definició de les taules del programa**. Les taules base son aquelles que constantment utilitzem i no volem repetir, (usuaris, perfils, mòduls...)

```
delimiter $$

CREATE TABLE IF NOT EXISTS AssignacioUsuarisGrups(
    LoginUsuari varchar(50) NOT NULL,
    IDGrup varchar(50) NOT NULL,
    PRIMARY KEY CLUSTERED (LoginUsuari ASC, IDGrup ASC)
) ENGINE=InnoDB AUTO_INCREMENT=88 DEFAULT CHARSET=utf8$$
```

- Un script per a les **relacions entre les taules base**. I un altre script per a les **relacions de les taules del programa**

```
ALTER TABLE Usuaris
  ADD CONSTRAINT FK_Usuaris_Pestanyes
  FOREIGN KEY (IDPestanyaDefecte)
  REFERENCES Pestanyes(IDPestanya)
  ON DELETE NO ACTION
  ON UPDATE NO ACTION
, ADD INDEX FK_Usuaris_Pestanyes_Idx(IDPestanyaDefecte ASC);
```

- Un script per a les **vistes**

- El més important: Un script individual per a cada funció de cada entitat del programa. Usualment CÀRREGA, BAIXA i MODIFICACIÓ/ALTA

Script per a la càrrega d'una entitat X:

En MySQL

```
drop procedure IF EXISTS PE_EntitatX_Carrega;

delimiter $$

CREATE PROCEDURE PE_EntitatX_Carrega
(pTipus VarChar(20), pID int)

Begin
if pTipus = null Then
        Set pTipus = '';
End If;
if pID = null Then
        Set pID = 0;
End If;

Set pTipus = upper(pTipus);

If pTipus = 'una_funcio' Then
        SELECT camp1, camp2, campn FROM taula Where ID = pID;
End If;
End$$

delimiter ;
```

L'script ha de començar esborrant el procedure de la BBDD, perquè si tenim la BBDD sotmesa al control de codi, aquest és l'únic que val.

Usualment el procediment per la càrrega espera dos paràmetres:

- pTipus
- pID

pTipus indica el mètode de càrrega que es vol emprar. Per exemple:

- Si pTipus = "" significa que s'espera el registre identificat amb el paràmetre pID, i que aquest no pot ser null
- Si pTipus = "TOT" significa que pID ha de ser null i que s'esperen tots els registres de l'entitat.
- Si pTipus = "ENTITAT2" significa que pID identifica un registre d'una tercera entitat, (entitat2), i que s'espera una llista de registres coincidents amb aquest filtre

pID identifica el registre que es vol carregar, o bé si ve informat a null significa que s'espera un llistat de tots els registres

Script per a l'esborrat de l'entitat X

És una solució semblant al cas de l'escript de càrrega.

En MySQL

```
drop procedure IF EXISTS PE_EntitatX_Esborra;

delimiter $$

CREATE PROCEDURE PE_EntitatX_Esborra
        (pID)

Begin
        if pID = null Then
                Set pID = '';
        End If;

        If pID = '' Then
                DELETE FROM ENTITAT WHERE ID = pID;
        End If;
End$$

delimiter ;
```

Usualment aquests procediments reben un paràmetre, (pID), que no pot ser null ni buit.

Script per la modificació/alta de l'entitat X

Aquest procediment empra una estructura XML com a paràmetre. D'aquesta forma tenim dues aventatges:

- No tenim una llista inacabable i difícil de gestionar de paràmetres del procediment
- Utilitzem les aventatges de toDataSet i toXML que ens proporciona el Model

Per a SQL Server 2005 o superior:

```
if exists (select * from dbo.sysobjects where id = object_id(N'[dbo].[PE_EntitatX_Salva]')
and OBJECTPROPERTY(id, N'IsProcedure') = 1)
drop procedure [dbo].[PE_EntitatX_Salva]
GO

SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE PROCEDURE [dbo].[PE_EntitatX_Salva]
        @pXML text

As Begin
        Set NoCount On

        Declare @idoc int

        If NOT @pXML Is NULL Begin
                Exec sp_xml_preparedocument @idoc Output, @pXML
```

```
                    SELECT * INTO #tmp
                    FROM OpenXml (@idoc, '/node arrel/node fill, 2 )
                            With ( camp1 int, camp2 bit, camp3 VarChar(50) )


                    If (SELECT Count(*) FROM #tmp) > 0 Begin
                            Declare @cCamp1 int
                            Declare @cCamp2 bit
                            Declare @cCamp3 VarChar(50)

                            DECLARE entitatX_Cursor CURSOR FOR
                            SELECT Camp1, Camp2, Camp3
                            FROM #tmp

                            OPEN entitatX_Cursor

                            FETCH NEXT FROM entitatX_Cursor
                            INTO @cCamp1, @cCamp2, @cCamp3

                            WHILE @@FETCH_STATUS = 0
                            BEGIN
                                    If (SELECT COUNT(*) FROM EntitatX WHERE ID =
@cID) > 0 Begin
                                            Delete EntitatX Where ID = @cID
                                    End

                                    INSERT INTO EntitatX (Camp1, Camp2, Camp3)
                                    VALUES (@cCamp1, @cCamp2, @cCamp3)

                                    FETCH NEXT FROM entitatX_Cursor
                                    INTO @cCamp1, @cCamp2, @cCamp3
                            END

                            CLOSE entitatX_Cursor
                            DEALLOCATE entitatX_Cursor
                    End
            End
    End
```

Com a nota, el sistema te una forma una mica “millorable” de distingir si el registre existeix, (i fa un update), o no exiteix, (i fa un insert. Aquí si existeix l’esborra per a que el resultat sigui sempre un insert. Això és molt millorable.

El sistema permet, (i això també val per a la solució en MySQL), guardar diversos registres alhora, ja que, com que li estem passant una estructura XML, aquesta pot tenir informació de diverses files. Per exemple:

```
<usuaris>
        <usuari>
                <camp1>dada 1</camp1>
                <camp2>dada 2</camp2>
        </usuari>
        <usuari>
                <camp1>dada 3</camp1>
```

```
        <camp2>dada 4</camp2>
    </usuari>
</usuaris>
```

Per a MySQL:

La solució per a MySQL és més “fina”. No empra cursors i per tant és més eficient.

```
drop procedure if exists PE_EntitatX_Salva;

delimiter $$

CREATE PROCEDURE PE_EntitatX_Salva
    (pXML text)

Begin
    declare i int default 1;
    declare nmax int default 0;

    If NOT pXML IS NULL Then
        set nmax := ExtractValue(pXML, 'count(//node)');

        while i <= nmax do
            set @cCamp1 := ExtractValue(pXML, '//node[$i]/Camp1');
            set @cCamp2 := ExtractValue(pXML, '//node[$i]/Camp2');
            set @cCamp3 := ExtractValue(pXML, '//node[$i]/Camp3);


            If (SELECT COUNT(*) FROM EntitatX WHERE ID = @cID) > 0
Then
                UPDATE EntitatX Set Camp1 = @Camp1 WHERE ID
= @cID;
            Else
                INSERT INTO EntitatX (Camp1, Camp2, Camp3)
VALUES (@cCamp1, @cCamp2, @cCamp3);
            End If;

            set i = i + 1;
        End while;
    End If;
End$$

delimiter ;
```

Altres qüestions sobre els procediments emmagatzemats

- Res de instruccions SQL en codi, excepte en casos molt concrets
- El nom dels procediments han de seguir una estructura determinada i sempre la mateixa. Per exemple:
  PE_EntitatX_[CARREGA | ESBORRA | SALVA]

- Es millor precedir d'un prefix tots els procediments, de forma que apareguin agrupats a la BBDD. Vigileu amb prefixos del tipus "SP" o altres que puguin tenir un ús especial al SGBD

- En el que a qüestions de càrrega i salvatatge es refereix. Aprofitar les possibilitats d'scripting per portar la lògica al script. SQL permet moltes eines per a seleccionar registres mitjançant filtres. En la messura del possible, no fem scripts que retornen tot i després fem filtres sobre codi

06. Model MVC

- Controladors
  - DBManager <Abstract>
  - CtrlClasseModel1
  - CtrlClasseModel2
- Model
  - Model <abstract>
  - ClasseModel1
  - ClasseModel2
- Vista
  - FrmFuncionalitat1
  - FrmFuncionalitat2

Sigui com sigui: **Sempre de la mateixa manera**. Un cop decidida l'estructura, aquesta s'ha de respectar. Tant en les carpetes, com en els noms de classes, mètodes, ús de interfaces, etc

06.1. Comprovacions bàsiques sobre el Model

1. Comprova **SEMPRE** si instàncies passades per paràmetre estan a null. Posteriorment comprova SEMPRE que allò clau que defineix la instància no estigui buit:

```
Public void load(Maquina m) {
    If (m != null) {
            If (m.codi != string.Empty) {
                    // Ara puc fer la feina
            }
    }
}
```

2. Descarrega sempre els recursos
   Maquina m = new Maquina();
   ....

```
m.dispose();
m = null;
```

3. Les propietats que son vincle a altres classes: Carrega-les només quan les necessitis:

```
public PerfilUsuari perfilUsuari {
    get {
        if (mPerfilUsuari == null) {
            mPerfilUsuari = new
            PerfilUsuari(Convert.ToString(mPropietats["IDPerfi
            l"]));
            try {
                mPerfilUsuari.carrega();
            } catch (Exception ex) {
                mPerfilUsuari = null;
            }
        }

        return mPerfilUsuari;
    }
```

D'aquesta forma no es produeix un efecte de saturació de la memòria al treballar amb molts objectes del model

4.

07. Empra Interfaces

Indispensables:

IAbstractModel

Determina si un objecte ha canviat alguna de les seves propietats des de la seva instanciació.
bool getHasChanged();

Determina si la instancia actual es igual a la passada per paràmetre.
bool Equals(object pO);

Retorna una cadena amb les propietats de la instáncia en XML.
string toXML(string pStructureName, string pDataUnitName);

Retorna una estructura DataSet amb els valors de les propietats de la instancia actual.
DataSet toDataSet(string pDataSetName, string pTableName);

Accepta com a vàlids els canvis fets sobre propietats de l'objecte.
void acceptChanges();

Cancel·la els canvis fets sobre l'objecte.
void cancelChanges();

IIdentified

Retorna la identificació de l'objecte de forma unívoca a la BBDD.
object[] getIdentification();

Retorna el contingut de la propietat descriptiva de l'objecte.
string getDescription();

ILoadable

Carrega l'objecte del model a partir de les dades de BD.
void load();

IModel

Implementa: ILoadable, ISaveable, IErasable, ICloneable, IDisposable i IIdentified

Determina si l'objecte existeix a la BBDD
bool getDBExist();

ISaveable

Mètode per fer persistent el contingut d'un objecte.
void save();

IClonable

IDisposable

De la BBDD:

IDBManager

Implementa: IDisposable

Propietat TimeOut
int getTimeout();
void setTimeout(int pTimeout);

Informació de la connexió.
string getConnectionInfo();

Propietat Nom de la base de dades
string getBDName();

Executa un procediment emmagatzemat.
DataTable executeProc(string pProcName, string pCursorDevolutionName);

Executa un procediment emmagatzemat.
DataTable executeProc(string pProcName, string pCursorDevolutionName, Controladors.DBParameter[] pParam);

Executa un procediment emmagatzemat sense retorn de resultats.
void executeProc(string pProcName);

Executa un procediment emmagatzemat sense retorn de resultats.
void executeProc(string pProcName, Controladors.DBParameter[] pParam);

Inicia una transacció per la connexió actual
void TransactionBegin();

Accepta la transacció
void TransactionAccept();

Cancel·la la transacció.
void TransactionCancel();

Tanca la transacció.
void TransactionEnd();

IDBParameter

Nom del paràmetre.
string getName();

Valor inicial del paràmetre.
object getValue();

Guarda un valor de devolució.
void setValue(object pValue);

Tipus de dada del paràmetre.
DbType getType();

Direcció d'E/S del paràmetre.
System.Data.ParameterDirection getDirection();

Mida màxima del contingut.
int getSize();

Altres:

IErasable

Elimina l'objecte de la BBDD a partir de les seves propietats clau.
void erase();

Determina si la instància actual es esborrable.
bool isErasable();

IRefreshable

Refresca la informació d'un objeto de domini.
void refresh();

ILockable

ISearchable

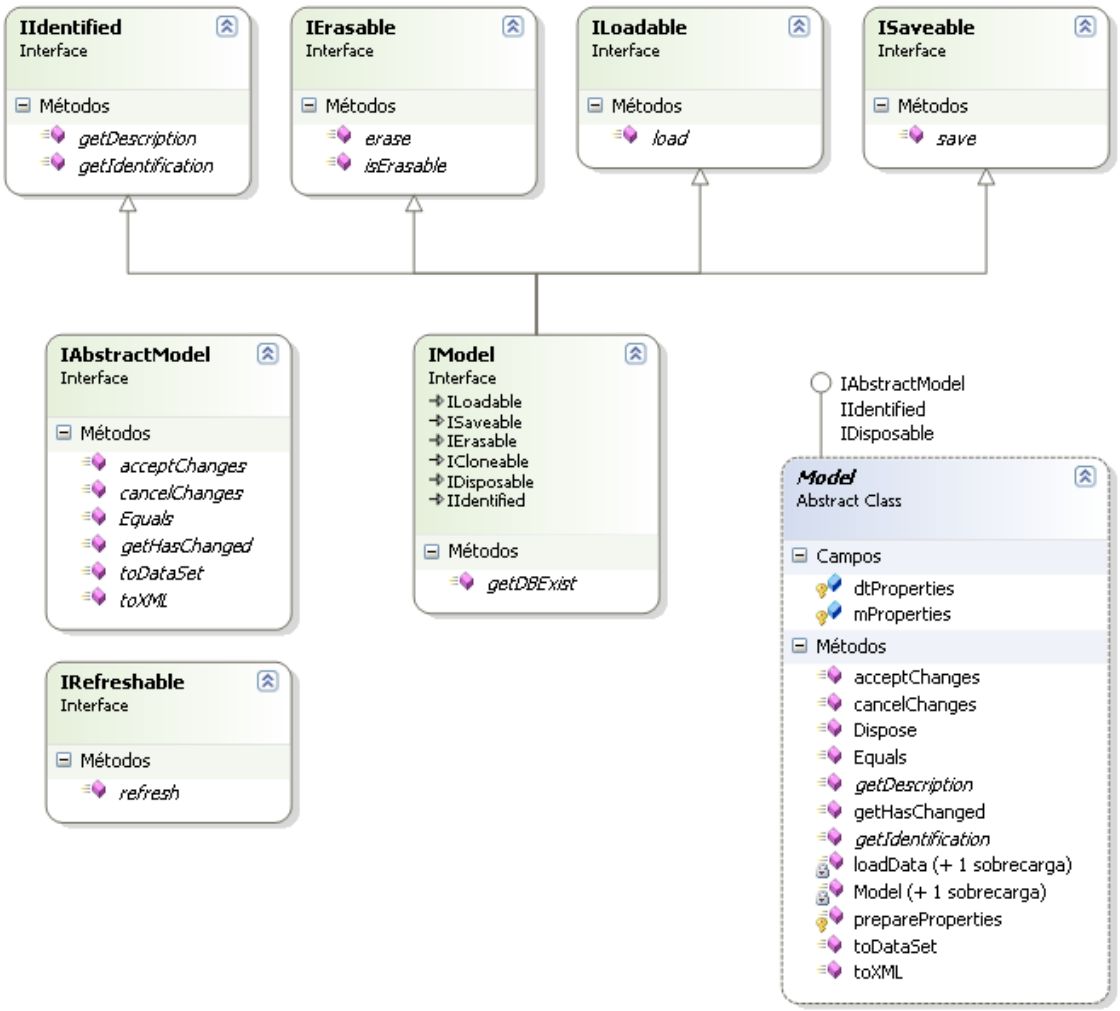
IPrintable

07.1. Com implementa les interfaces el domini

Tota classe de domini ha d'implementar els mateixos mètodes. Per a fer-ho, el millor és crear una classe genèrica de domini abstracta

Model

Que implementa IAbstractModel, IIdentified i IDisposable. Això obliga a implementar els mètodes que figuren a les interfaces descrites, i també a aquelles a les que tenen relació. Tots aquesta mètodes i propietats son comuns a totes les classes del model i, generalment, s'implementen sempre de la mateixa manera. Cal crear una estructura de dades i de implementació que ens estalvii repetir sempre els mateixos mètodes que s'0implementen igual.



Totes les classes del model hereten de Model, i implementen IModel i, opcionalment, IRefreshable, ILockable, ISearchable, IPrintable, etc

Amb la implementació de IModel, la classe del model es veu obligada a implementar allò que és comú en qualsevol classe del model, però que s'implementa diferent:

- La persistència, (ILoadable, ISaveable i IErasable)
- La identificació unívoca, (IIdentified)
- Els suports per la gestió, (IDisposable, ICloneable...)

IDisposable acaba estant a la classe del model que implementem i a la clase abstracta que ens serveix de suport, (Model). Això és així perquè IDisposable es crida recursívament per assegurar així que tanquem tots els recursos abans de tancar la classe. Això és molt important. Abans de fer classe = null, **hem d'assegurar-nos que tanquem tots els recursos** dels que feia ús. Una cosa tant senzilla com aquesta provoca:

- Millora en el rendiment
- Ens protegeix davant d'errors no controlats

```
Classe hereta de Model
Public override void Dispose() {
        Base.Dispose();
```

```
        // Tanca recursos de la instància
}
```

08. Els controladors

Hi ha dues menes de controladors:
- Controladors de la BBDD
- Controladors de la relació entre Model i Vista

08.1. Els controladors de la BBDD

Els controladors de la BD implementen les característiques particulars de cada SGBD, i **ens proporcionen una interfície comuna de treball que ens aïlla d'aquestes particularitats**. No programem per a Oracle o per a SQLServer. Programem de forma genèrica per a SGBD, i **deleguem les particularitats** al controlador.

La interfície comuna ens la proporciona una Interface, (IDBManager). Tots els gestors del model, que precisen ús de la BBDD, han d'emprar els mètodes definits aquí, i no altres.

DBManager és la classe abstracta que implementa els mètodes comuns, i on decideix quina classe de gestió de BD particular utilitza, (per exemple DBManager_SQLServer o DBManager_MySQL...).. Una forma de determinar quina classe especialitza és emprant el constructor:

```
private Interfaces.IDBManager gd;

protected DBManager (TipusGestorDadesEnum pGestor) {
        try {
                switch (pGestor) {
                        case TipusGestorDadesEnum.MySQL: // Gestor de IBM
                        AS400.
                                gd = new DBManager _MySQL();
                                break;
                        default:
                                gd = new DBManager _SQLServer();
                                break;
                }
        } catch (Exception ex) {
                throw ex;
        }
}
```

Finalment, tota classe gestora del Model, (Ctrl_ClasseX), implementa la classe abstracta DBManager.

Sempre que sigui possible empra procediments emmagatzemats en el SGBD

08.2. Els controladors del model

Les funcions principals de càrrega, desament, esborrat, cerca, etc... Sempre es delegen en un controlador. **No s'implementen directament en la classe del model**. La classe del model crida els mètodes del controlador per a fer la tasca encomanada. Per exemple:

```
Model:
Public void load() {
        cursor = controladorModel1.getDades();
        setPropietat1(cursor.valor1);
        setPropietat2(cursor.valor2);
        ...
}

Controlador:
Private gestor = new DBManager();
Public cursor getDades() {
        Cursor res = new Cursor();

        parametre1 = new DBParameter(clau1);
        res = gestor.executeProc("ProcedimentBD", parametre1);

        return res;
}
```

En la mesura del possible, els controladors reben com a paràmetres instàncies del model, i no dades. Per exemple:
Correcte: Public void setMaquina(Maquina m)
No correcte: Public void setMaquina(inti d, string nom)

08.3. Els constructors del model

Els constructors d'una classe estan per fer-los servir. Hi ha diversos escenaris en que els constructors son útils en alguna mesura:

- Volem impedir que es pugui instanciar una classe "buida": Fem privat el constructor sense paràmetres: private void Maquina()
- Volem fer autocàrrega. Quan s'instancii una determinada classe aquesta ja es carregui de la BD: Hem de crear un constructor amb l'identificador:
  ```
  Public Maquina(int id): this() {
      Load(id);
  }
  ```
- Els constructors SEMPRE criden a un constructor més senzill. Amb "this" o amb "base"
- El constructor més senzill sempre inicialitza valors per defecte de la classe
- Estranyament una classe del model implementa singleton
  - Sempre la classe del domini, al constructor més senzill instancia el seu controlador

08.4. Sobrecàrrega de mètodes

Sempre que sigui necessari fer una mateixa acció, però aquesta depengui dels valors d'entrada, hem d'emprar sobrecàrrega de mètodes. Per exemple, en les classes del model estem obligats a implementar ILoadable, que porta el mètode load().
Suposem que tenim necessitat de carregar una instància a partir del seu id. Podríem fer el següent:

Public void loadid(inti d)

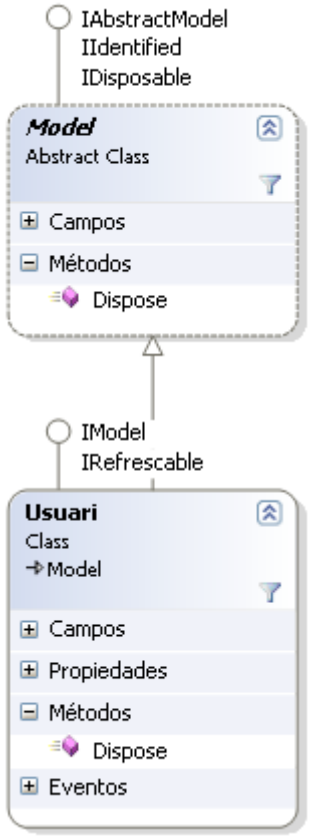
Això no és incorrecte, però portat a l'extrem, (i en els desenvolupaments tot acaba sempre anant a l'extrem), porta a confusions, a usos incorrectes d'alguns mètodes i a errors no controlats.

La forma correcta seria:

Public void load(inti d)

D'aquesta forma, **TOT l'equip** de desenvolupament sap una cosa molt important i molt bàsica: "La única forma vàlida de carregar una instància del model és amb el mètode load". L'IDE, al escriure load, ens oferirà:
Load()
Load(inti d)

Hi ha una derivada en aquesta qüestió de la sobrecàrrega, que es basa en el fet que, alhora estem fent ús de classes abstractes, que ja implementen funcionalitat comuna, en la personalització que estem fent podem reimplementar un mètode o afegir noves signatures a aquest mètode. Per exemple, l'us de Dispose entre la classe abstracta del model i la especialització del model



Usuari:

```
public new void Dispose() {
    base.Dispose();

    // Cierra el gestor.
    gestor.Dispose();
    gestor = null;
}
```

Per a fer aquesta sobrecàrrega es necessita precedir el mètode de la classe filla per la directiva “new” o “virtual i override”

09. Tractar en tot moment les excepcions

Tractar les excepcions és la diferència entre una aplicació robusta i una aplicació amb descrèdit. Les excepcions, de forma general s’han de tractar aplicant les premisses bàsiques següents:

1. Excepcions a nivell de mètode
2. Centralitzar la canalització de les excepcions
3. Emprar excepcions pròpies
4. Gestionar les excepcions a partir d’un gestor

09.1. Excepcions a nivell de mètode

Tot mètode ha d'estar englobat en un try...catch. Totes les excepcions possibles que es puguin produir al mètode s'han de tractar de forma especial. Evitar en el possible l'ús genèric de Exception

```
Try {
        // El contingut del mètode
} catch (ExcepcioEspecial1 ex1) {
        // Excepció personalitzada segons els possibles riscos del mètode. Per exemple: IOException
} catch  (ExcepcioEspecial2 ex2) {
        // Un altre. Tota possible excepció s'ha de gestionar de forma personalitzada
} catch (Exception ex) {
        // Qualsevol altra excepció no controlada
} finally {
        // Mètodes que s'han d'executar encara que es produeixi una excepció
        // Especial atenció amb el fet de tancar recursos oberts pel mètode, i destruir accions i objectes que puguin haver quedat a mitjes
}
```

09.2. Centralitzar la canalització de les excepcions

La forma de centralitzar és pujar les excepcions cap al nivell més superior possible. Usualment el formulari o el mètode principal que crida al programa.

```
Try {
        ...
Catch (Exception ex) {
        throw new Excepcions.ExcepcioGestorDadesNoAccessible(this, ex);
}
```

Per centralitzar les excepcions cal fer dues coses:

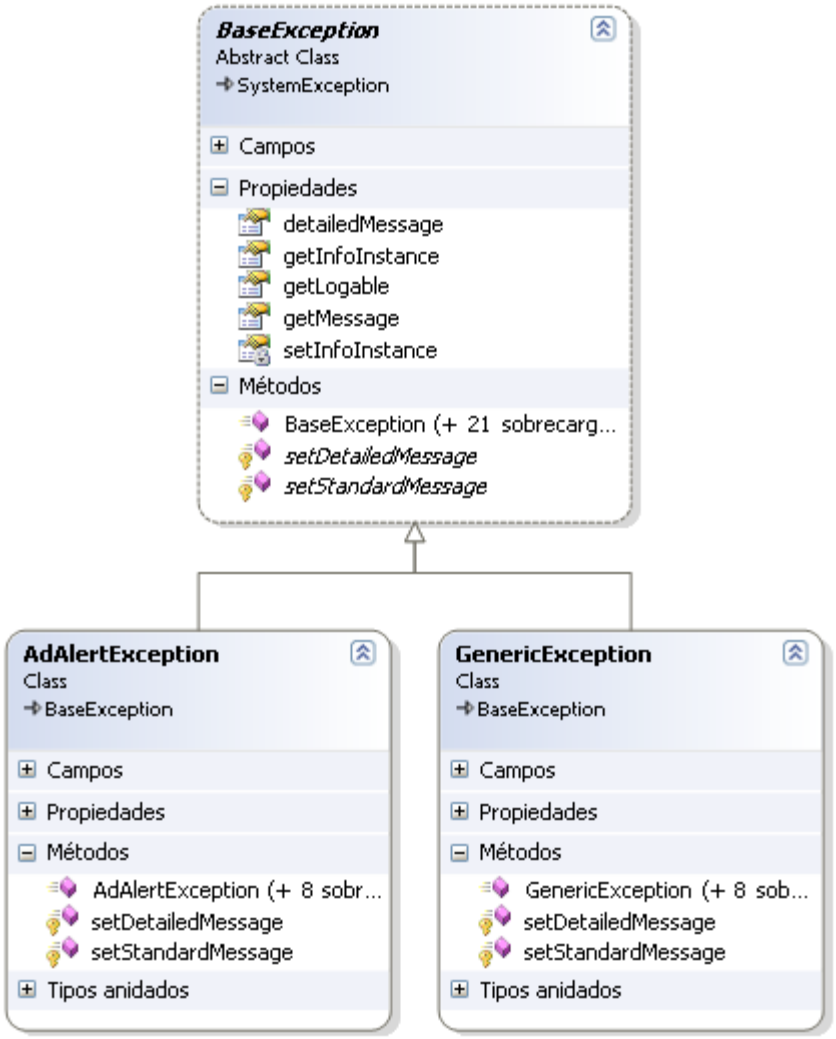
1. **Pujar-les**
2. **Personalitzar-les**. La forma que tenim posteriorment de gestionar les excepcions, passa per "traduir-les" a un format que després puguem entendre. Una forma és creant un arbre d'excepcions propi. Amb excepcions per tipus, o excepcions per grups.

L'excepció va pujant per les diferents classes i mètodes que cursen l'execució de l'aplicació, fins arribar a un punt inicial. Si pel camí no cal afegir més informació, simplement farem "throw ex". Si pel camí podem afegir més informació, farem:

```
throw new ExcepcióPersonalitzada (this, ex);
```

De forma que, en arribar al punt inicial, el que tenim és un arbre d'excepcions, que podem mostrar de forma resumida o detallada, emprant per a mostrar l'error, un gestor d'excepcions que sap gestionar les excepcions personalitzades que em creat.

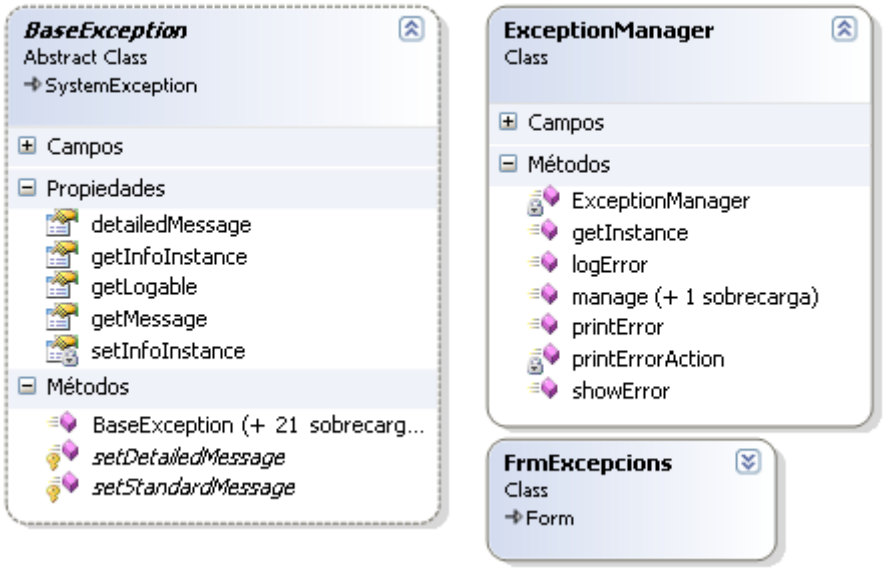
09.3. Emprar excepcions pròpies



Per crear una estructura d'excepcions personalitzada cal crear objectes Exception que proporcioni la informació que necessitem. Per a fer-ho és convenient crear una classe abstracta que hereti les funcionalitats bàsiques de Exception, (o de SystemException). Posteriorment crearem tantes excepcions personalitzades com necessitem. En l'exemple de la imatge només creem 2:

- **GenericException**: Gestiona les excepcions comunes sobre funcions bàsiques del model, com per exemple errors al carregar, al desar, al clonar, etc
- ModelException: Correspondria a les excepcions pròpies de l'aplicació. Generalment porten el nom de l'aplicació + "exception". Serien excepcions del tipus "No s'ha pogut guardar un client". "No es tenen permisos per a veure una gestió", etc.

09.4. Gestionar les excepcions a partir d'un gestor



Un cop hem de mostrar l'excepció, i proposar a l'usuari accions per a resoldre el problema, és convenient utilitzar un gestor de excepcions propi. Aquest gestor automatitza les accions bàsiques, tals com mostrar l'excepció, imprimir-la, enviar-la per mail, vincular-la amb el sistema de log i traçabilitat, etc.