Mock Objects

Index

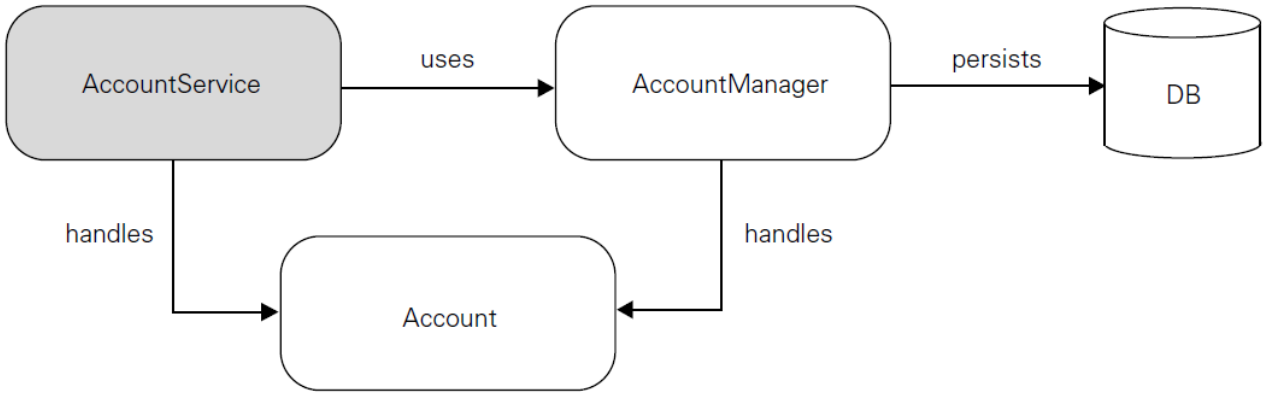
- Introducció
- Exemple
- *Refactoring* i *Torjan Horses*
- *Best practices*

Index

- **Introducció**
- Exemple
- *Refactoring* i *Torjan Horses*
- *Best practices*

Introducció

- Suposem que tenim una arquitectura com la següent



`AccountService`: Ofereix serveis relacionats amb `Account`.

`Account`: Compte bancari.

`AccountManager`: Gestiona les dades d'un `Account` utilitzant una DB.

- Quins problemes pràctics creieu que poden apareixer si volem fer unit-testing dels mètodes de la classe `AccountService`?
  - Necessitem que les classes que utilitza `AccountService` estiguin implementades?
  - Necessitem que hi hagi una DB en funcionament?

Resposta: NO! ➔ ***Mock objects*!**

Introducció

- Definició: "***Mock object*** is an object created to stand in for an object that your code will be collaborating with. Your code can call methods on the mock object, which will deliver results as set up by your tests". *Vincent Massol "JUnit in Action"*

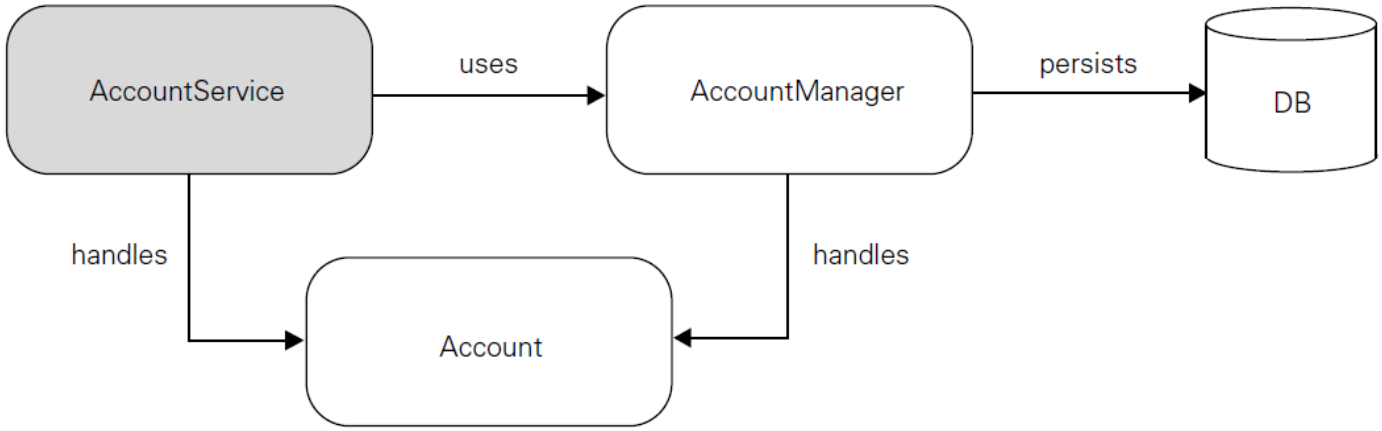
- Caracteristiques:
  - **Substitueixen** els **objectes** amb els que interactua el SUT *(Subject under Test)*
  - Ofereixen una **capa d'aillament**
  - No implementen **cap lògica** de **funcionament** (excepte la **mínima imprescindible** que necessita el cas de test)
  - Són **capes buides** que **proveeixen metodes** amb els que interactua el SUT
  - Són una "mentida", una façana

Index

- Introducció
- **Exemple**
- *Refactoring* i *Torjan Horses*
- *Best practices*

Exemple

- Volem desenvolupar codi de test (d'unitat) pel mètode `AccountService.transfer()`



- Per quina classe necessitem un mock object?
  - `AccountManager`: És qui gestiona i necessita una DB que funcioni

Exemple

- Declaració de la classe `AccountManager`

```
public interface AccountManager
{
    Account findAccountForUser(String userId);
    void updateAccount(Account account);
}
```

La seva implementació interactua amb la DB

- Definició de la classe `AccountService`

```
public class AccountService
{
    private AccountManager accountManager;
    public void setAccountManager(AccountManager manager)
    {
        this.accountManager = manager;
    }
    public void transfer(String senderId, String beneficiaryId, long amount)
    {
        Account sender = this.accountManager.findAccountForUser(senderId);
        Account beneficiary = this.accountManager.findAccountForUser(beneficiaryId);
        sender.debit(amount);
        beneficiary.credit(amount);
        this.accountManager.updateAccount(sender);
        this.accountManager.updateAccount(beneficiary);
    }
}
```

Copia una referencia a un objecte extern de tipus AccountManager

Exemple

- Definició del mock `MockAccountManager`

```
public class MockAccountManager implements AccountManager
{
```

La declarem filla de la classe que volem **"imitar"**

```
   private Hashtable accounts = new Hashtable();
   public void addAccount(String userId, Account account)
   {
      this.accounts.put(userId, account);
   }
```

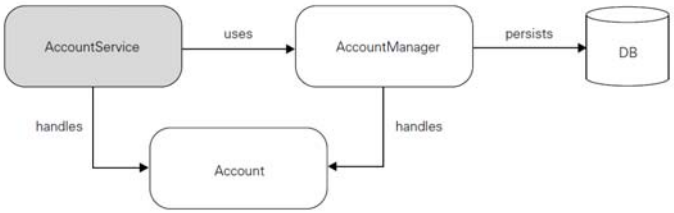
Definim i inicialitzem una hashtable per **"imitar"** el funcionament de la DB

```
   public Account findAccountForUser(String userId)
   {
      return (Account) this.accounts.get(userId);
   }
```

Simulem una query a la DB

```
   public void updateAccount(Account account)
   {
      // do nothing
   }
}
```

No fem res!!! No ho necessitem doncs no hi ha DB real



Exemple

- Codi de test

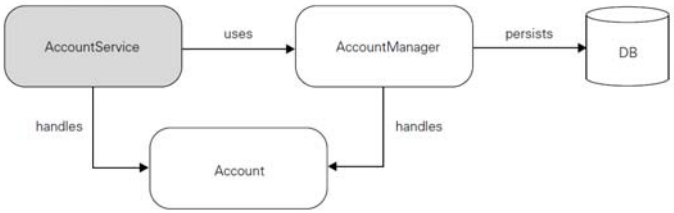
```
public class TestAccountService extends TestCase
{
    public void testTransferOk()
    {
        MockAccountManager mockAccountManager = new MockAccountManager();

        Account senderAccount = new Account("1", 200);
        Account beneficiaryAccount = new Account("2", 100);
        mockAccountManager.addAccount("1", senderAccount);
        mockAccountManager.addAccount("2", beneficiaryAccount);

        AccountService accountService = new AccountService();
        accountService.setAccountManager(mockAccountManager);

        accountService.transfer("1", "2", 50);

        assertEquals(150, senderAccount.getBalance());
        assertEquals(150, beneficiaryAccount.getBalance());
    }
}
```

Declaració *mock object*

Setup del mock object

Creació i setup de `AccountService`

Prova i validació de `AccountService`



Index

- Introducció
- Exemple
- ***Refactoring* i *Torjan Horses***
- *Best practices*

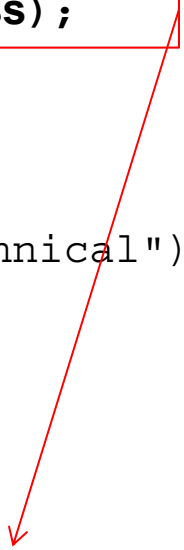
Refactoring

- Abans hem utilitzat els mock objects segons el pattern *Inversion of Control* (IOC):
  - Externalització de la creació dels objectes necessaris (no es creen dins la classe)
  - Enviament a la classe a través de mètodes (constructors, setters, …).

Refactoring

- Suposem el següent codi

```
public class DefaultAccountManager implements AccountManager
{
   private static final Log LOGGER = LogFactory.getLog(AccountManager.class);

   public Account findAccountForUser(String userId)
   {
     LOGGER.debug("Getting account for user [" + userId + "]");
     ResourceBundle bundle = PropertyResourceBundle.getBundle("technical");

     String sql = bundle.getString("FIND_ACCOUNT_FOR_USER");
     // Some code logic to load a user account using JDBC
     [...]
   }

   [...]

}
```



Objecte intern. NO es pot utilitzar un altre!

Si (p.ex. durant el test) volem que LOGGER no faci res, no podem!

Refactoring

- Utilitzem *mock objects* i *Inversion of Control* IOC
  - Això ens permetrà fer:

Creació i configuració de LOGGER com a *mock*

```
public void testFindAccountByUser()
{
   MockLog logger = new MockLog();
   MockConfiguration configuration = new MockConfiguration();
   configuration.setSQL("SELECT * [...]");

   DefaultAccountManager am = new DefaultAccountManager(logger,configuration);

   Account account = am.findAccountForUser("1234");
   // Perform asserts here
   [...]

}
```

Creació i configuració de SUT enviant-li el *mock*

Creació i test del SUT

Trojan Horse

- *Expectation*: Caracterísitca d'un mock que comprova si el comportament d'un SUT és el correcte
  - Exemples: A través d'un *mock* podem comprovar si
    - `AccountManager` crida al metode `close` cada cop que acaba de fer una consulta a una base de dades
    - Un mètode obre un fitxer en disc i al final el tanca
    - (en C++) alocació i desalocació de memòria
    - …

Index

- Introducció
- Exemple
- *Refactoring* i *Torjan Horses*
- ***Best practices***

Best practices

- Quan utilitzar mock objects
  - L'objecte real te un **comportament no determinista**
  - L'objecte real és **dificil** de **configurar**
  - L'objecte real és **dificil** de **portar** a un **estat** (*network error*)
  - L'objecte real és **lent**
  - L'objecte real és una **API** o **UI**
  - L'objecte real **encara no existeix**!

Best practices

- *Best practices*
  - **NO** escriure **logica de funcionament** dins un mock (veure següent punt)
  - **Només** ha de **fer** el que el codi de **test li diu** que ha de fer
  - S'han de **poder generar facilment**
  - Són facilment "trencables" (*breakable*). **NO** necessiten ***testing***!

Best practices

- Quins son els **avantatges** dels *mock objects*?
  - Fer **test sense** esperar **SW extern**
  - **Evitar** problemes o **efectes secundaris** d'objectes **externs**
  - Permeten fer **refactoring**
  - Permeten **comprobar patterns** de **comportament** del SUT correctes

Exercici

- Suposem que volem desenvolupar un SW que ens calculi diferents estadistiques sobre els alumnes de la Escola d'Enginyeria. Tenim un servidor de base de dades amb informació sobre tots els alumnes de l'escola. Per simplificar l'exemple suposem que la informació que es guarda és una taula amb atributs (`NIU, assignatura, Nteo, NPract, NFinal`) que estan guardats com cadenes de caracters (`String`).

  Suposeu que teniu la declaració de la classe que realitza la gestió de la base de dades:

```
public interface DB
{
   public boolean connect();
   public String[][] query(String q);
   public boolean close();
}
```

  Nota: per convertir una `String` a `double`, podeu utilitzar `Double.parseDouble(String);`

- Implementeu una classe (desenvolupada sota el paradigma TDD) amb un mètode que permeti calcular el tant per cent d'alumnes aprovats, suspesos i no presentats d'una nota d'una assignatura concreta. Aquesta classe utilitza la classe DB per obtenir les dades de la base de dades. Suposeu que aquesta classe ve definida com:

```
public class EstadistiquesEnginyeria
{
   public double PerCentAprovats(String Assignatura, String Nota);
   public double PerCentSuspesos(String Assignatura, String Nota);
   public double PerCentNoPresentats(String Assignatura, String Nota);
}
```

  on `Nota` agafa un dels valors "`NTeo`", "`NPract`" o "`NFinal`".