

Software Testing

Integration Testing

Limiti dello Unit Test

Non si può testare un modulo in isolamento se:

- comunica con il database
- comunica in rete
- comunica con altri moduli non ancora testati
- modifica database/file o altre fonti di dati
- non può essere lanciato in parallelo ad altri test
- bisogna effettuare diverse operazioni per lanciare il test
- ...

Testing di un modulo "non terminale": soluzione con driver e stub

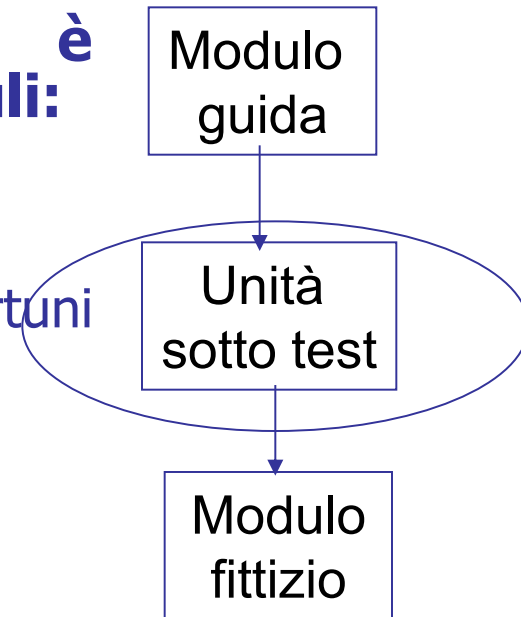
Per testare un modulo non terminale, è necessario costruire due tipologie di moduli:

Moduli guida (driver)

- invocano l'unità sotto test, inviandole opportuni valori, relativi al test case

Moduli fittizi (*stub*)

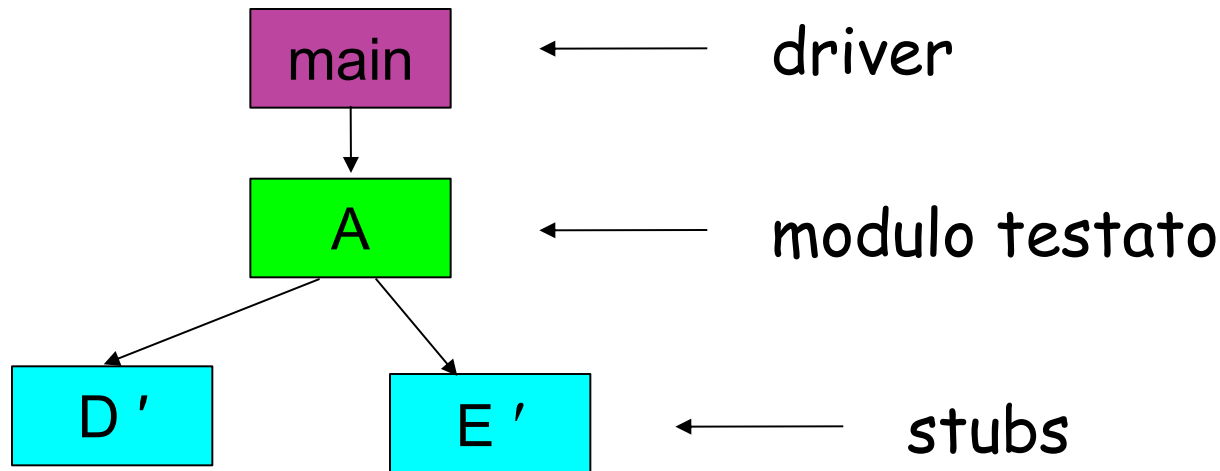
- sono invocati dall'unità sotto test;
- emulano il funzionamento della funzione chiamata rispetto al caso di test richiesto (tenendo conto delle specifiche della funzione chiamata)
 - *Quando la funzione chiamata viene realizzata e testata, si sostituisce lo stub con la funzione stessa*



Stub e Driver

I moduli testati hanno bisogno di essere chiamati (dai Driver)

I moduli chiamati devono essere sostituiti da altri (Stub)



Driver

- **Un modulo driver deve sostituire in tutto e per tutto il/i moduli chiamanti il modulo da testare**
 - Un metodo TestCase sotto Junit può implementare un driver
- **Il modulo driver deve:**
 - Settare tutti i valori delle risorse e fonti dati utilizzate dal modulo da testare
 - In linguaggi object oriented, costruire l'oggetto il cui metodo è sotto test
 - Avviare il metodo da testare

Stub

- **Uno stub è una funzione fittizia la cui correttezza è vera per ipotesi**
 - Esempio, se stiamo testando una funzione *prod_scal(v1,v2)* che richiama una funzione *prodotto(a,b)* ma non abbiamo ancora realizzato tale funzione
 - Nel metodo driver scriviamo il codice per eseguire alcuni casi di test
 - *Ad esempio chiamiamo prod_scal([2,4],[4,7])*
 - Il metodo stub potrà essere scritto così:

```
int prodotto (int a, int b){  
    if (a==2 && b==4) return 8;  
    if (a==4 && b==7) return 28;  
}
```
 - La correttezza di questo metodo stub è data per ipotesi
 - Ovviamente per poter impostare tale testing, bisognerà avere precise informazioni sul comportamento interno richiesto al modulo da testare

Stub

- **Il termine Stub è utilizzato, più genericamente, per indicare un metodo fittizio, non ancora implementato o la cui implementazione sia, volutamente, incompleta**
 - Spesso gli stub vengono messi nel codice semplicemente come promemoria dei metodi ancora da realizzare oppure per consentire la compilazione del codice prima possibile
 - Lo stub ha il compito di riprodurre il comportamento del modulo che sostituisce unicamente nei casi di test previsti dai driver realizzati
 - Lo stub può essere scritto sulla base di una conoscenza 'black box' del modulo da emulare
 - *Gli stub consentono di testare un modulo prima che i moduli da cui esso dipenda sono stati realizzati*

Stub

- **Uno Stub può sostituire efficacemente una funzione**
 - Ad ogni caso di test deve corrispondere una diversa istanza dello stub (oppure un ramo diverso dello stub)
- **Uno Stub non sostituisce efficacemente una classe**
 - Non può gestire lo stato (valori degli attributi) di una classe tra due chiamate dello stub
 - Non può gestire il testing di una sequenza di interazioni

Dipendenze

- **Come si fa a sapere da quali moduli *dipende* l'esecuzione di un dato modulo da testare?**
 - *Il grafo delle dipendenze (Dependency Graph) è un grafo i cui nodi rappresentano moduli (eventualmente classi, metodi, package) e i cui archi, orientati, rappresentano relazioni di dipendenza tra i moduli (ad esempio causate dall'esistenza di chiamate di metodo, utilizzo di oggetti, utilizzo di attributi)*

Dependency graph evaluation

- **Il grafo delle dipendenze può essere parzialmente (totalmente in alcuni casi particolari) ricavato dall'analisi del codice sorgente dell'applicazione**
- **Parecchie estensioni eclipse sono in grado di valutare il dependency graph**
 - stan4j
 - Eclipse Metrics
 - jDepend
 - eDepend

Eclipse Metrics

- **Eclipse Metrics è una estensione di eclipse che verrà utilizzata anche per calcolare automaticamente metriche relative al codice sorgente**
 - Un tutorial completo sul suo utilizzo è all'indirizzo <http://metrics.sourceforge.net/>
 - La home page del progetto, open source, è all'indirizzo <http://sourceforge.net/projects/metrics/>

Eclipse Metrics

- **Nell'ambito del testing di integrazione Eclipse metrics consente di:**
 - Disegnare il grafo delle dipendenze di un progetto a partire dal suo codice sorgente
 - Ottenere un ordine topologico dei package, da quelli indipendenti a quelli con maggiore dipendenza
 - Limite: non fornisce indicazioni (nel dependency graph), a livello di classi e metodi

stan4j

- **Strumento molto più potente per la valutazione di dependency graph e altre metriche**
 - Free solo per utilizzi su sistemi di limitate dimensioni
 - Un tutorial sul suo utilizzo è all'indirizzo:
<http://stan4j.com/>
 - Scaricabile da:
<http://stan4j.com/general/download.html>
- **Disegna il grafo delle dipendenze anche a livello più dettagliato, delle classi e dei metodi**

Testing dell'integrazione di due moduli

- **Una volta testate tutte le unità, è necessario procedere al testing di integrazione per valutare se esse funzionano correttamente nel loro complesso**
 - Se a chiama b, e sia a che b hanno superato i test di unità, non è detto che l'insieme di a e b soddisfi tutti i test previsti di a
 - *Possibili problemi potrebbero essere legati, ad esempio, a combinazioni negli input di b che chi ha testato b non ha previsto ma che possono comparire come possibili valori di chiamata in a*
 - Per testare a e b (con a che chiama b) potrebbe essere sufficiente utilizzare i test progettati per a
 - *In pratica, si sostituisce, nei test progettati per a, il vero modulo b al posto del suo stub (o mock)*

Strategie per il testing di integrazione

- **Il testing di integrazione consiste nel considerare insiemi via via più grandi di moduli, fino ad ottenere l'insieme completo**
- **In che modo decidiamo l'ordine di integrazione?**
 - Due strategie "estreme": top-down e bottom-up

Testing d'integrazione bottom-up

- **Si parte dai moduli che non hanno dipendenze (nodi senza archi uscenti nel grafo delle dipendenze)**
- **Si integra ognuno di questi moduli con uno di quelli che lo chiama e si testa la coppia**
- **Si ridisegna il grafo delle dipendenze avendo sostituito i due moduli integrati con un unico modulo**
- **Si ripete iterativamente il procedimento fino ad ottenere un grafo con un unico nodo**
 - Questo procedimento vale in assenza di cicli nel grafo.

Testing d'integrazione top-down

- **Si parte dai moduli che non dipendono da alcun altro modulo (nodi senza archi entranti nel grafo delle dipendenze)**
- **Si integra ognuno di questi moduli con uno di quelli chiamati e si testa la coppia**
- **Si ridisegna il grafo delle dipendenze avendo sostituito i due moduli integrati con un unico modulo**
- **Si ripete iterativamente il procedimento fino ad ottenere un grafo con un unico nodo**
 - Questo procedimento vale in assenza di cicli nel grafo.

Risoluzione dei cicli

Esistono diverse tecniche possibili:

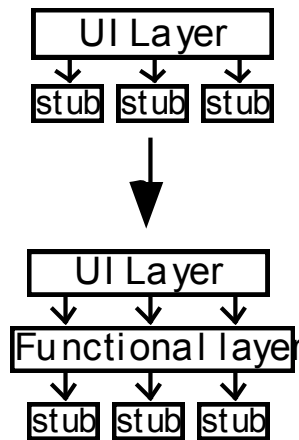
- **Ridurre i cicli includenti più moduli a cicli di due soli moduli, dopo integrazioni successive di moduli tra loro non ciclici**
 - $a \rightarrow b \rightarrow c \rightarrow a$ diventa $a \rightarrow bc \rightarrow a$, poi $abc \rightarrow a$, infine abc
- **«Aprire» i cicli utilizzando degli stub**
 - Ad esempio, in presenza di un ciclo $a \rightarrow b \rightarrow c \rightarrow a$
 - *si sdoppia il modulo a creando un aStub*
 - *Si integrano i moduli come $a \rightarrow b \rightarrow c \rightarrow aStub$ (non ciclici)*
 - *Si sostituisce a ad aStub testando tutto il ciclo*

Strategie d'integrazione

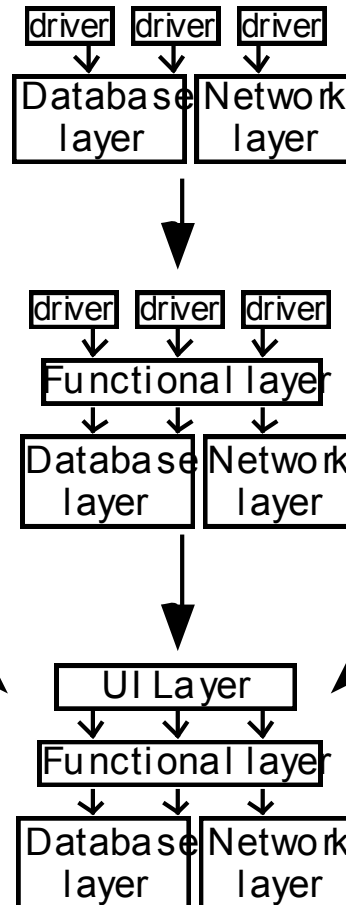
- **E'abbastanza semplice la scelta di una strategia precisa per software progettato a livelli (layer)**
- **Spesso si utilizzano tecniche miste (sandwich) nel quale strategie top-down e bottom-up sono alternate**
- **Altre strategie portano a integrare prima i metodi più fortemente accoppiati, in modo da ridurre il più velocemente possibile la complessità del grafo delle dipendenze**
- **Spesso la strategia d'integrazione dipende anche dalla gerarchia aziendale e dall'organizzazione delle risorse umane**

Strategie per il testing di integrazione

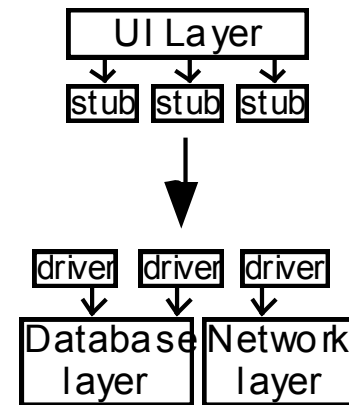
Top-down testing



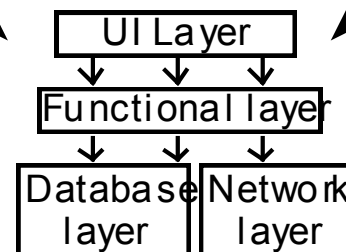
Bottom-up testing



Sandwich testing

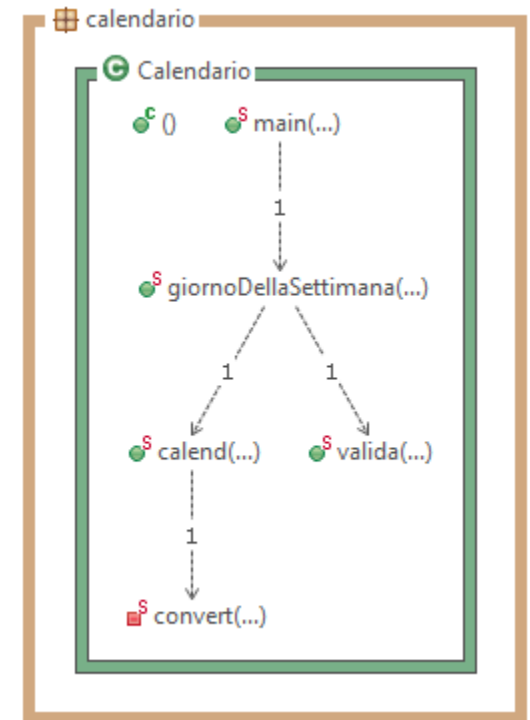


Fully integrated system



Esempio di testing di integrazione con JUnit

- **Consideriamo un'unica classe Calendario con i seguenti metodi:**
 - main legge da linea di comando giorno, mese (stringa) e anno
 - giornoDellaSettimana converte il mese in input in un intero, controlla se la data è valida e eventualmente chiama calend
 - valida controlla se la data è valida
 - calend calcola il giorno della settimana (numerico)
 - convert converte in stringa il giorno della settimana risultante



Codice esempio: giornoDellaSettimana

```
public static String giornoDellaSettimana(int d, String ms, int a)
{
    int m=0;
    if (ms.equals("gennaio"))
        m=1;
    else if (ms.equals("febbraio"))
        m=2;
    else if (ms.equals("marzo"))
        m=3;
    else if (ms.equals("aprile"))
        m=4;
    else if (ms.equals("maggio"))
        m=5;
    else if (ms.equals("giugno"))
        m=6;
    else if (ms.equals("luglio"))
        m=7;
    else if (ms.equals("agosto"))
        m=8;
    else if (ms.equals("settembre"))
        m=9;
    else if (ms.equals("ottobre"))
        m=10;
    else if (ms.equals("novembre"))
        m=11;
    else if (ms.equals("dicembre"))
        m=12;
    if ((m>0) && valida(d,m,a))
        return calend(d,m,a);
    else
        return "Errore";
}
```

Codice esempio: valida

```
public static boolean valida(int d, int m, int a) {  
    if (d<1 || d>31 || m==0 || a<=1582)  
        return false;  
    Boolean bisestile= (a%4==0);  
    if (bisestile && a%100==0 && a%400!=0)  
        bisestile=false;  
    if ((m==2 && d>29) || (m==2 && d==29 &&  
        !bisestile))  
        return false;  
    if ((m==4 || m==6 || m==9 || m==11) && d>30)  
        return false;  
    return true;  
}
```

Codice esempio: calend

```
public static String calend(int d, int m, int a)
{
  if (m<=2)
  {
    m = m + 12;
    a--;
  };
  int f1 = a / 4;
  int f2 = a / 100;
  int f3 = a / 400;
  int f4 = (int) (2 * m + (.6 * (m + 1)));
  int f5 = a + d + 1;
  int x = f1 - f2 + f3 + f4 + f5;
  int k = x / 7;
  int n = x - k * 7;
  return convert(n);
}
```

Codice esempio: convert

```
public static String convert(int n) {  
    if (n==1)  
        return "Lunedì";  
    else if (n==2)  
        return "Martedì";  
    else if (n==3)  
        return "Mercoledì";  
    else if (n==4)  
        return "Giovedì";  
    else if (n==5)  
        return "Venerdì";  
    else if (n==6)  
        return "Sabato";  
    else if (n==0)  
        return "Domenica";  
    else return "Errore";  
}
```

Codice esempio: main

```
public static String main(String[] args) {  
    if (args.length==3){  
        int giorno=Integer.parseInt(args[0]);  
        String mese=args[1];  
        int anno=Integer.parseInt(args[2]);  
        String giornoDS=new String(giornoDellaSettimana(giorno,mese,anno));  
        System.out.println(giornoDS);  
        return giornoDS;  
    }  
    else  
        return "";  
}
```

Esempio di testing bottom-up 1/2

- **Cominciamo testando convert**

`@Test`

```
public void testConvert1() {  
    assertEquals("Domenica", Calendario.convert(0));  
}
```

...

- **Continuiamo con calend**

`@Test`

```
public void testCalend1() {  
    assertEquals("Domenica", Calendario.calend(24,  
        4, 2011));  
}
```

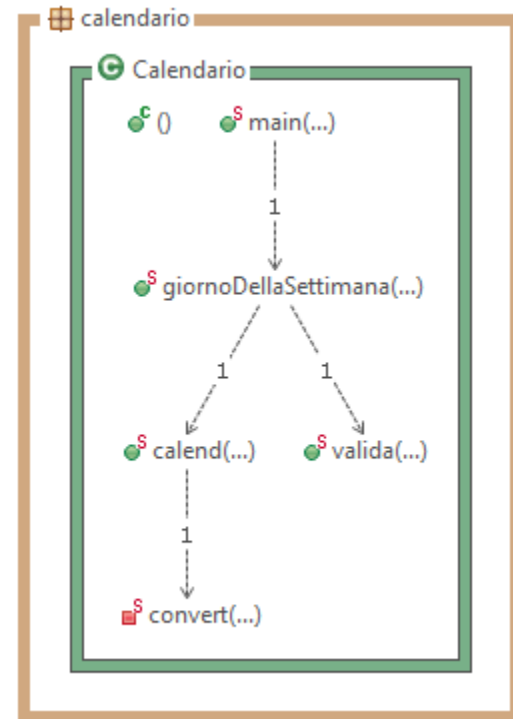
...

- **Poi valida:**

`@Test`

```
public void testValida1() {  
    assertTrue(Calendario.valida(24, 4, 2011));  
}
```

...



Esempio di testing bottom-up 2/2

- **Poi giornoDellaSettimana:**

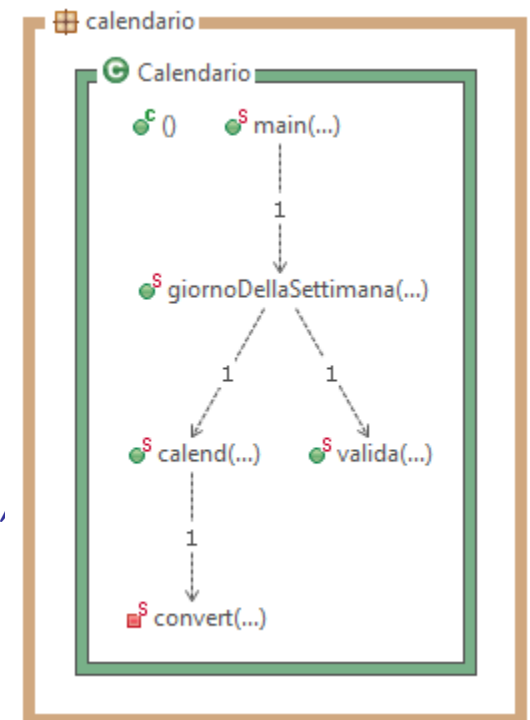
```
@Test
public void testGiornoDellaSettimana1() {
    assertEquals("Domenica", Calendario.giornoDellaSettimana(24,
        "aprile", 2011));
}
```

...

- **Infine il test del main:**

```
@Test
public void testMain1() {
    String a[]=new String[3];
    a[0]="24";
    a[1]="aprile";
    a[2]="2011";
    assertEquals("Domenica", Calendario.main(a)),
}
```

...

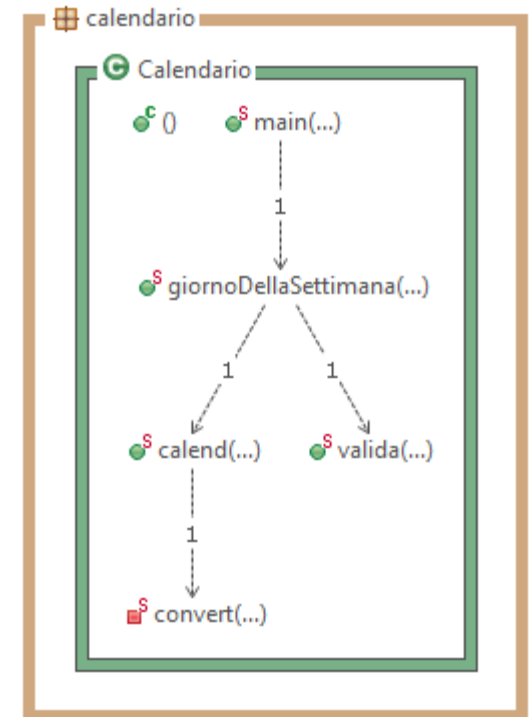


Esempio di testing top-down 1/2

- **Per testare la classe main da sola, vediamo che è necessario solo uno stub per giornoDellaSettimana**

```
public static String giornoDellaSettimana(int
    d, String ms, int a){
//STUB
if (d==24 && ms.equals("aprile") && a==2011)
    return "Domenica"; //TC1
else if (d==32 && ms.equals("aprile") &&
    a==2011) return ""; //TC2
return "";
}
```

- **Possiamo, poi, eseguire gli stessi test progettati per il main nel bottom-up**



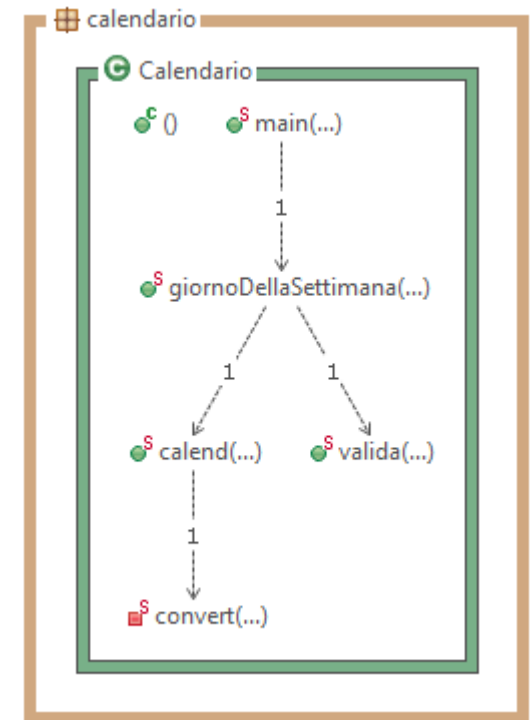
Esempio di testing top-down 2/2

- Per testare la classe `giornoDellaSettimana` sono necessari stub sia per `calend` che per `valida`

```
public static boolean valida(int d, int m, int a) {  
    //STUB  
    if (d==24 && m==4 && a==2011) return true;  
    else if (d==29 && m==2 && a==2012) return true;  
    else if (d==32 && m==4 && a==2011) return  
        false;  
    return false;  
}
```

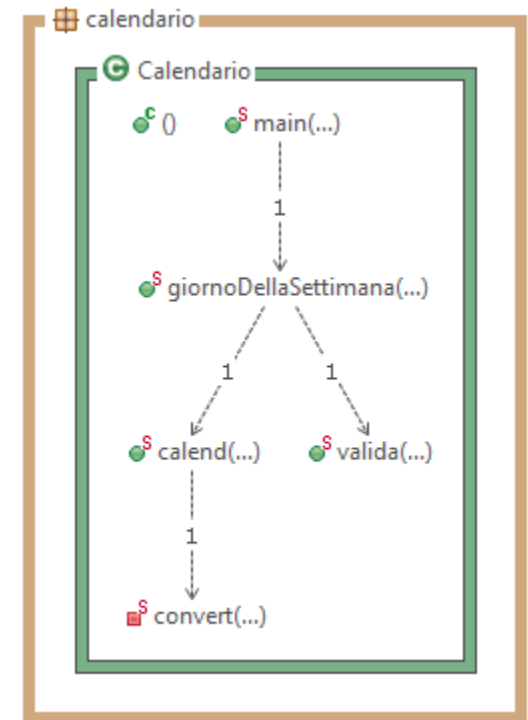
```
public static String calend(int d, int m, int a){  
    // STUB  
    if (d==24 && m==4 && a==2011) return  
        "Domenica";  
    else if (d==32 && m==4 && a==2011) return  
        "Errore";  
    else return "";  
}
```

- A questo punto riusciamo i test di `calend`, e così via



Altri casi di test

- **Se, ad esempio, la classe calend fosse stata la prima ad essere realizzata, sarebbero necessari**
 - Ad esempio per ragioni di prototipazione, poiché è l'unica classe con complessità derivanti dalle conoscenze di astronomia
- **Un driver che si comporti come giornoDellaSettimana**
 - Può essere il metodo di test JUnit che pensammo nelle strategie top-down e bottom-up
- **Uno stub che imiti convert nei casi previsti dal driver**
 - Può essere lo stub che pensammo nel testing bottom-up



Test Doubles

In generale, un test double è una implementazione alternativa e sostitutiva di una interfaccia o di una classe che non potrebbe essere utilizzata in un test perchè:

- Troppo lenta
- Non (ancora) disponibile
- Dipendente da qualcos'altro che non è (ancora) disponibile
- Troppo difficile da istanziare e configurare per un test
 - Solo nel caso particolare di una funzione, un suo test double può essere uno stub

Tassonomia: Mock Object, Fake, e Stub

Type of mock	Description
Stubs	Stubs are essentially the simplest possible implementation of a given interface you can think of. For example, stubs' methods typically return hardcoded, meaningless values.
Fakes	Fakes are a degree more sophisticated than stubs in that they can be considered an alternative implementation of the interface. In other words, a fake looks like a duck and walks like a duck even though it isn't a real duck. In contrast, a stub only looks like a duck.
Mocks	Mocks can be considered even more sophisticated in terms of their implementation, because they incorporate assertions for verifying expected collaboration with other objects during a test. Depending on the implementation of a mock, it can be set up either to return hardcoded values or to provide a fake implementation of the logic. Mocks are typically generated dynamically with frameworks and libraries, such as EasyMock, but they can also be implemented by hand.

Tratto da: Test Driven - Practical TDD and Acceptance TDD
for Java Developers (Lasse Koskela)- Manning Ed. 2008

Mock-Objects

- **Una precisa emulazione del comportamento delle classi non ancora implementate è ottenibile con i mock object**
- **Un mock object è una simulazione di un oggetto reale.**
- **Implementa l'interfaccia dell'oggetto da simulare ed ha il suo stesso comportamento.**
- **Possono fornire una risposta pre-impostata.**
- **Possono verificare se l'oggetto che li usa lo fa correttamente.**
- **Utilissimi per testare unità senza legarsi ad oggetti esterni.**

Differenze fra Stub e Mock

I Mock-Objects non sono Stub! [Fowler]

- <http://martinfowler.com/articles/mocksArentStubs.html>

In genere lo stub è molto più semplice di un Mock-Object

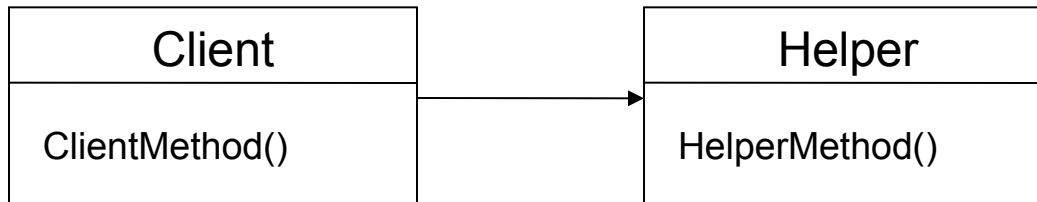
- Gli stub forniscono risposte preconfezionate (con valori prefissati) a chiamate fatte durante il test, senza rispondere di solito a nulla che sia al di fuori di ciò che è previsto per il test.

I mock hanno implementazioni più sofisticate che consentono di verificare il comportamento dell'unità testata (e non solo lo stato)

- verificando ad esempio le collaborazioni avute con altri oggetti ed il relativo ordine di esecuzione
- Possono contenere asserzioni, riguardanti aspetti utili al debugging
- Mock già realizzati saranno fondamentali nei test che coinvolgono classi di libreria

Testing di Classi con Mock

Es.: La classe Client (da testare) usa i metodi di Helper



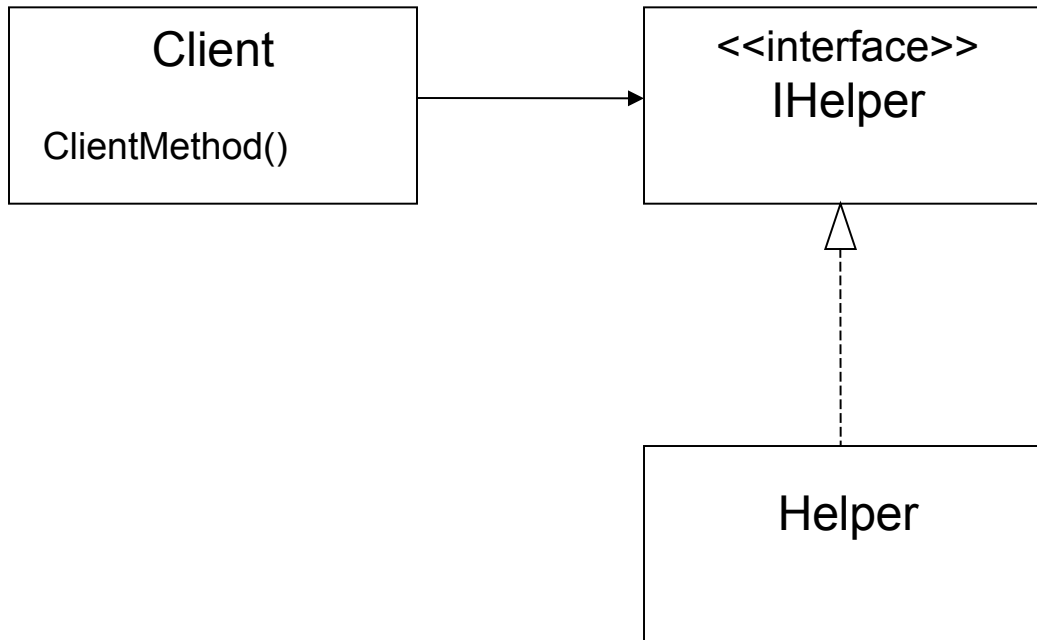
Ma la classe Helper non può essere usata perché:

- Non è ancora disponibile
- Vogliamo controllare direttamente ciò che Helper restituisce a Client

Vogliamo costruire un Test Double di Helper

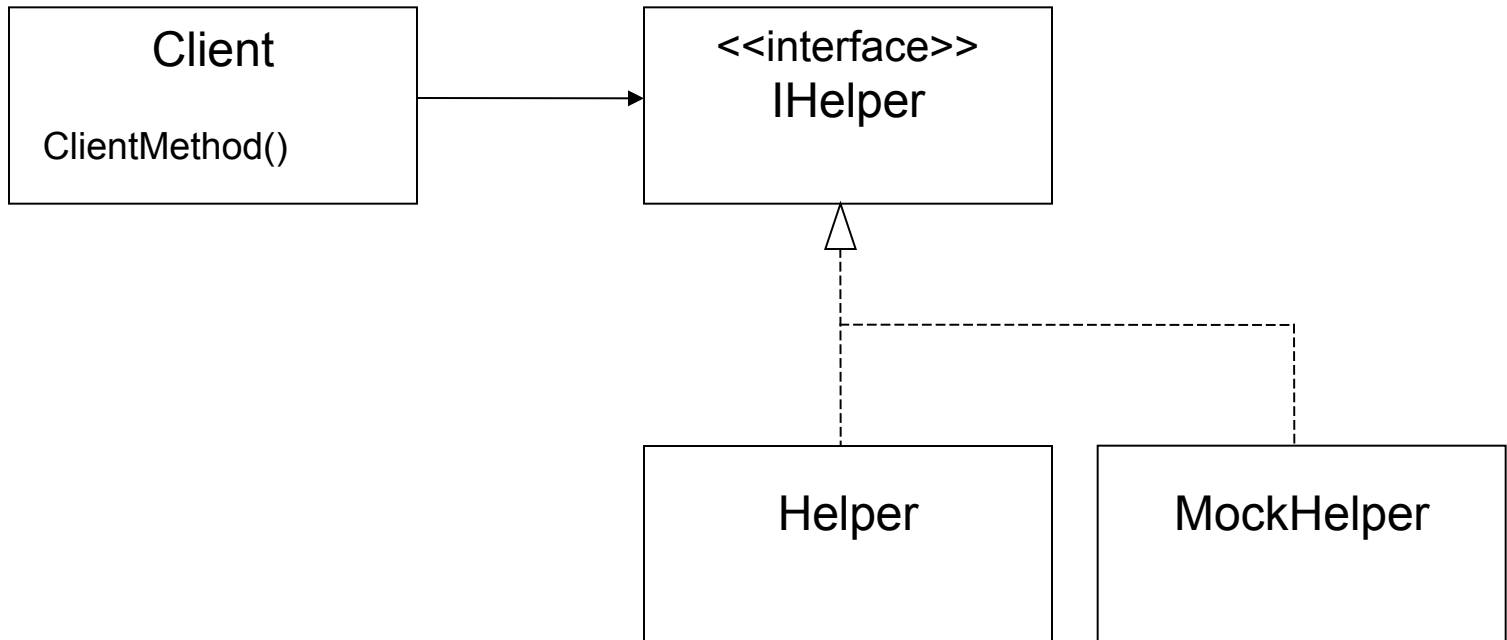
La soluzione usando i Mock-Object

Si estrae l'Interfaccia IHelper...



La soluzione usando i Mock-Object

Il MockObject implementa l'Interfaccia IHelper



Un esempio di semplice Mock

```
public class MockHelper implements IHelper
{
    public MockHelper ( )
    {
    }

    public Object helperMethod( Object aParameter )
    {
        Object result = null;
        if ( ! aParameter.toString().equals("expected" )
        {
            throw new IllegalArgumentException(
                "Unexpected parameter: " + aParameter.toString() );
        }
        result = new String("reply");
        return result;
    }
}
```

Utilizzo di MockHelper

Una classe di test che testi la classe Client utilizzando MockHelper anziché mock dovrà:

- **Istanziare un oggetto di MockHelper**
- **Passarlo al costruttore di Client o ad un metodo setHelper di Client**

In questo modo, la classe Helper originale potrà in futuro essere integrata senza la necessità di modificare o distruggere alcunchè

Esempio: classe di test

- Per rendere eseguibile questo complesso caso di test funzionale è stato sufficiente scrivere PricingServiceTestDouble anziché PricingService
 - Da notare che la sostituzione tra oggetto reale e suo doppione avviene solo e soltanto nel caso di test, non nell'originale PricingService

```
public class OrderProcessorTest {  
    @Test  
    public void testOrderProcessorWithMockObject() throws Exception {  
        float initialBalance = 100.0f;  
        float listPrice = 30.0f;  
        float discount = 10.0f;  
        float expectedBalance = (initialBalance - (listPrice * (1 - discount/100)));  
        Customer customer = new Customer(initialBalance);  
        Product product = new Product("TDD in Action", listPrice);  
        OrderProcessor processor = new OrderProcessor();  
        PricingService service = new PricingServiceTestDouble(discount);  
        processor.setPricingService(service);  
        processor.process(new Order(customer, product));  
        assertEquals(expectedBalance, customer.getBalance(), 0.001f);  
    }  
}
```

Esempio (con sola estensione anziché Interface)

Doppione della classe Pricing Service:

- Estende la classe (interfaccia) che deve sostituire
- Ridefinisce i metodi di cui abbiamo bisogno per eseguire i nostri test
 - In questo caso consente di accedere all'attributo discount
- Il metodo reale getDiscountPercentage avrebbe letto il valore dal database

```
public class PricingServiceTestDouble extends PricingService {  
    private float discount;  
    public PricingServiceTestDouble(float discount) {  
        this.discount = discount;  
    }  
    public float getDiscountPercentage(Customer c, Product p) {  
        return discount;  
    }  
}
```

Sviluppo di Mock

- **In genere i Mock sono in grado di fare controlli sul comportamento dell'oggetto testato e sulle sue interazioni con altri oggetti.**
- **Lo sviluppo di Mock Objects è supportata da diversi frameworks, o librerie (come JMock o EasyMock in Java).**
 - Spesso sono disponibili librerie di mock già pronti corrispondenti ad oggetti di libreria

EasyMock

É un Framework per creare mock objects a run time.

Usa le Java reflection per creare una classe mock object che implementa una certa interfaccia.

É un progetto open source project ospitato su SourceForge:

<http://www.easymock.org>

Esempio: class Portfolio

```
package esempioeasymock;
import java.util.ArrayList;
import java.util.List;

public class Portfolio {
    private String name;
    private StockMarket stockMarket;
    private List<Stock> stocks = new
        ArrayList<Stock>();

    public Double getTotalValue() {
        Double value = 0.0;
        for (Stock stock : this.stocks) {
            value += (stockMarket.getPrice(stock.getName()) *
                stock.getQuantity());
        }
        return value;
    }

    public String getName() {return name;}

    public void setName(String name) {this.name = name;}

    public List<Stock> getStocks() {return stocks;}

    public void setStocks(List<Stock> stocks)
        { this.stocks = stocks; }

    public void addStock(Stock stock) { stocks.add(stock);}

    public StockMarket getStockMarket()
        {return stockMarket;}

    public void setStockMarket(StockMarket stockMarket)
        {this.stockMarket = stockMarket;}
}

Classe Portfolio:
Gestisce un insieme di azioni (Stock) ed è in grado di
calcolarne il valore totale sommando i singoli valori
```

Esempio: classe Stock e interface StockMarket

```
package esempioeasymock;
```

```
public class Stock {  
    private String name;  
    private int quantity;  
  
    public Stock(String name, int quantity) {  
        this.name = name;  
        this.quantity = quantity;  
    }  
  
    public String getName() {return name;}  
    public void setName(String name)  
        {this.name = name;}  
    public int getQuantity() {return quantity;}  
    public void setQuantity(int quantity)  
        {this.quantity = quantity;}  
}
```

```
package esempioeasymock;
```

```
public interface StockMarket {  
    public Double getPrice(String stockName);  
}
```

- **Stock rappresenta un azione.**
- **StockMarket è un'interfaccia della quale non è (ancora) disponibile alcuna implementazione**
- **Vogliamo testare i principali metodi di Portfolio sostituendo a StockMarket (che non esiste ancora) un mock creato con EasyMock**

Esempio: testPortfolio

```
package esempioeasymotests;
import junit.framework.TestCase;
import org.easymock.EasyMock;
import org.junit.Before;
import org.junit.Test;
import esempioeasymock.*

public class PortfolioTest extends TestCase {
    private Portfolio portfolio;
    private StockMarket marketMock;

    @Before
    public void setUp() {
        portfolio = new Portfolio();
        portfolio.setName("Veera's portfol");
        marketMock =
            EasyMock.createMock(StockMarket.class);
        portfolio.setStockMarket(marketMock);
    }
}
```

```
@Test
public void testGetTotalValue() {
    EasyMock.expect(marketMock.getPrice("EBAY")).and
        Return(42.00);
    EasyMock.replay(marketMock);

    Stock ebayStock = new Stock("EBAY", 2);
    portfolio.addStock(ebayStock);
    assertEquals(84.00, portfolio.getTotalValue());
}
}
```

- marketMock è una istanza di un'implementazione mock di StockMarket creata a run-time da EasyMock
- Con il metodo EasyMock.expect viene dichiarato uno stub del comportamento di getPrice, intendendo che quando getPrice verrà chiamato con parametro di valore pari a «EBAY» dovrà essere restituito il valore 42 (metodo andReturn)
- Con il metodo replay vengono creati effettivamente i metodi stub le cui caratteristiche sono state dichiarate con gli expect
- Il resto del metodo di test è identico a quello che sarebbe stato se non fosse esistito alcun Mock

Esempio di testing di interazioni con JDBC

Classe di cui vogliamo creare un Test Double

```
import javax.sql.*;
import java.sql.*;
import java.util.*;

public class JdbcPersonDao implements PersonDao {
    private DataSource datasource;

    public void setDatasource(DataSource datasource) {
        this.datasource = datasource;
    }

    public List<Person> findByLastname(String lastname) {
        try {
            Connection conn = datasource.getConnection();
            String sql = "SELECT * FROM people WHERE last_name = ?";
            PreparedStatement stmt = conn.prepareStatement(sql);
            stmt.setString(1, lastname);

            ResultSet rset = stmt.executeQuery();

            List<Person> people = new ArrayList<Person>();
```

```
            while (rset.next()) {
                String firstName = rset.getString("first_name");
                String lastName = rset.getString("last_name");
                people.add(new Person(firstName, lastName));
            }
            rset.close();
            stmt.close();
            conn.close();
            return people;
        } catch (SQLException e) {throw new RuntimeException(e);}
    }
    // Other PersonDao methods not shown
}
```

- PersonDao è la classe Model che modella l'entità Persona
- JdbcPersonDao è la classe che implementa la connessione ad un database JDBC e, in particolare, il metodo findByLastName

Esempio di test JDBC 1/2

```
import static org.junit.Assert.*;
import static org.easymock.EasyMock.*;
import com.mockobjects.sql.*;
import org.junit.*;
import java.sql.*;
import javax.sql.*;
import java.util.*;

public class JdbcPersonDaoTest {
    @Test
    public void testFindByLastname() throws Exception {
        //Crea i mock e li collega
        DataSource datasource = createMock(DataSource.class);
        Connection connection = createMock(Connection.class);
        expect(datasource.getConnection()).andReturn(connection);
        String sql = "SELECT * FROM people WHERE last_name = ?";
        PreparedStatement stmt
        =createMock(PreparedStatement.class);
        expect(connection.prepareStatement(sql)).andReturn(stmt);
        stmt.setString(1, "Smith");
```

```
//Crea un mock del risultato
```

```
MockMultiRowResultSet resultset =new
MockMultiRowResultSet();
String[] columns = new String[] { "first_name", "last_name" };
resultset.setColumnNames(columns);
List<Person> smiths = createListOfPeopleWithLastname("Smith");
resultset.setupRows(asResultSetArray(smiths));
expect(stmt.executeQuery()).andReturn(resultset);
resultset.setExpectedCloseCalls(1);
stmt.close();
connection.close();
replay(datasource, connection, stmt);
```

Esempio di test JDBC 2/2

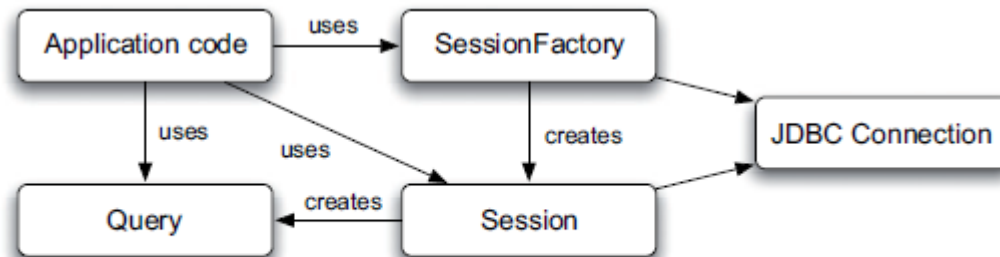
```
//testa il metodo dao.findByLastname di JdbcPersonDao
JdbcPersonDao dao = new JdbcPersonDao();
dao.setDatasource(datasource);
List<Person> people = dao.findByLastname("Smith");
assertEquals(smiths, people);
verify(datasource, connection, stmt);
resultset.verify();
}
```

```
private List<Person> createListOfPeopleWithLastname(String lastName) {
//genera la lista di risultati attesi
List<Person> expected = new ArrayList<Person>();
expected.add(new Person("Alice", lastName));
expected.add(new Person("Billy", lastName));
expected.add(new Person("Clark", lastName));
return expected;
}
```

```
private Object[][] asResultSetArray(List<Person> people) {
//Trasforma la lista in ResultSet
Object[][] array = new Object[people.size()][2];
for (int i = 0; i < array.length; i++) {
Person person = people.get(i);
array[i] = new Object[] {
person.getFirstName(),
person.getLastName() };
}
return array;
}
}
```

Hibernate

- Hibernate è un framework (API) che consente di inserire un ulteriore livello di virtualizzazione tra il software (Model nel caso MVC) e il database
- JDBC ad esempio consentiva la virtualizzazione del livello di accesso al database tramite una API che consentiva di interrogare tramite query SQL il database
- Hibernate invece consente di poter agire sui dati anche in forma di astrazioni Object Oriented
 - Hibernate implementa un linguaggio di query specifico: Hibernate Query Language (HQL)
 - Hibernate fornisce tre astrazioni fondamentali:
 - *SessionFactory*, che fornisce una *façade della sorgente dati*;
 - *Session*, che corrisponde ad una *connection di JDBC*, tramite la quale è possibile fare operazioni *CRUD sui dati*;
 - *Query*, che consente di fare *query più complesse*



Esempio di test per Hibernate

```
import static org.easymock.EasyMock.*;
import static org.junit.Assert.*;
import org.junit.*;
import java.util.*;
import org.hibernate.*;
import org.hibernate.classic.Session;

public class HibernatePersonDaoTest {
    private SessionFactory factory;
    private Session session;
    private Query query;

    @Before
    public void setUp() {
        //Crea mock per factory, session, query
        factory = createMock(SessionFactory.class);
        session = createMock(Session.class);
        query = createMock(Query.class);
    }
}
```

Si suppone che esista già una classe `HibernatePersonDao` creata da Hibernate per virtualizzare l'accesso ai dati dell'oggetto `Person`

```
@Test
public void testFindByLastname() throws Exception {
    String hql = "from Person p where p.lastname = :lastname";
    String name = "Smith";
    List<Person> theSmiths = new ArrayList<Person>();
    theSmiths.add(new Person("Alice", name));
    theSmiths.add(new Person("Billy", name));
    theSmiths.add(new Person("Clark", name));
    // Definisci esiti per ciò che necessita per la creazione di una query
    expect(factory.getCurrentSession()).andReturn(session);
    expect(session.createQuery(hql)).andReturn(query);
    expect(query.setParameter("lastname",name)).andReturn(query);
    expect(query.list()).andReturn(theSmiths);
    replay(factory, session, query);

    //Chiama il metodo da testare (dao.findByLastName) verificando che il risultato
    //sia lo stesso ottenibile con i mock
    HibernatePersonDao dao = new HibernatePersonDao();
    dao.setSessionFactory(factory);
    assertEquals(theSmiths, dao.findByLastname(name));
    verify(factory, session, query);
}
}
```

Ricapitolando Easy Mock

- **Le principali features di Easy Mock sono :**
 - Creazione dinamica di una classe mock:
 - `oggettomock = createMock(classe)`
 - Definizione dinamica di uno stub:
 - `expect(oggettomock.metododicuicrearestub.andReturn(valoredaritornare)`
 - Messa in funzione di tutti i mock e stub definiti:
 - `Replay(oggettomock1, oggettomock2, ...)`
 - Verifica (asserzione) dell'avvenuto corretto utilizzo degli oggetti mock:
 - `Verify (oggettomock1, oggettomock2, ...)`
-