Junit

Version 1.0.0

Module Eclipse

Les tests unitaires sous Eclipse avec Junit

30/07/2004

ITREC Gestion: 59, rue de Billancourt – BP 56 – 92105 BOULOGNE BILLANCOURT Cedex
Téléphone : 01 46 03 60 58 – Télécopieur : 01 46 03 61 81 - www.itrec.com

Création

Nom	Société	Fonction	Date
Dahan Julien	ITREC Gestion	Ingénieur d'étude et développement	30/07/2004

Modification

Nom	Société	Fonction	Date	Visa

Validation

Nom	Société	Fonction	Date	Visa
Martinez Frédéric	ITREC Gestion	Chef de projet		

Suivi des modifications

Date	Version	Descriptif
30/07/2004	1.00	Création du document.

Sommaire

INTRODUCTION

Le Framework ***JUnit*** permet de mettre en place des scénarios de test unitaires. On organise ainsi les tests en classes en créant une classe de test par classe à tester. ***JUnit*** permet ensuite d'exécuter en séquence les tests écrits par le développeur.

Ce framework est l'œuvre conjointe de Kent Beck (créateur de XP) et Erich Gamma (auteur des *Design Patterns*); il est gratuit et téléchargeable à partir du site junit.org. Il est écrit pour le langage Java, mais des frameworks gratuits inspirés de celui-ci sont également disponibles pour de nombreux autres langages : C++, Python...

Réaliser des tests présente un avantage certain :

- Analyse de petite granularité.
- Certitude d'engendrer peu de bugs.
- Non régression du code.
- Documente efficacement le code.

Ecrire efficacement des tests demande un peu de pratique, toutefois l'utilisation d'un framework de tests unitaires est essentielle à la constitution d'un code robuste; cela permet de placer le code en situation difficile et d'en améliorer la lisibilité.

Un test peut être écrit à différentes phases du développement. Le principe de **l'*eXtreme Programming*** invite le développeur à écrire son code *avant* même de commencer à coder ; certains pratiquent encore le **Pair Programming** ou programmation en binôme où l'on mène une discussion autour du développement à effectuer au cours des 30 minutes à venir. L'un des développeurs écrit les tests tandis que l'autre implémente le code. Les tests s'écrivent *en même temps* que le code ; il ne faut pas écrire tous les tests d'un coup avant le développement, ni les écrire après avoir terminé l'implémentation.

L'écriture des tests *avant* le code a plusieurs avantages :

- Affiner l'analyse en recensant les cas d'utilisation.
- Apporter l'ensemble des règles pré et post - conditions.
- Apporter l'ensemble des traitements de vérification de paramètres.
- Eviter d'écrire du code inutile : si les tests passent tous, il est inutile d'ajouter du code à la classe.

1. MISE EN ŒUVRE DE JUNIT SOUS ECLIPSE

1.1 IMPORTER LE PROJET JAVA DANS ECLIPSE

1.2 IMPORTER LE FICHIER JUNIT.JAR

- Click avec le bouton droit sur le nom du projet
- Sélectionner « propriétés »
- Dans l'arborescence de gauche sélectionner « Java Build Path »
- Dans l'onglet « Libraries » sélectionner « Add External JARs »
- Uiliser le browser pour trouver le fichier junit.jar
- Fermer la page des propriétés

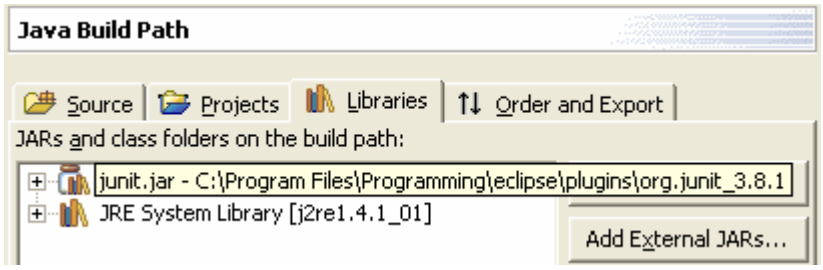

Figure - 1 Choisir « Add External JARs » et inclure le fichier junit.jar

1.3 CREER LA CLASSE DE TEST POUR LA CLASSE A TESTER

- Se positionner avec la souris sur le répertoire choisi pour spécifier l'emplacement de la classe de test.
- Fichier -> Nouveau -> Autres
- Dans l'arborescence de gauche faire dérouler le menu « Java » et sélectionner « Junit ».
- Dans la liste de droite sélectionner « TestCase » c'est à dire « scénario de test » (et non pas « suite de tests ») puis « suivant ».

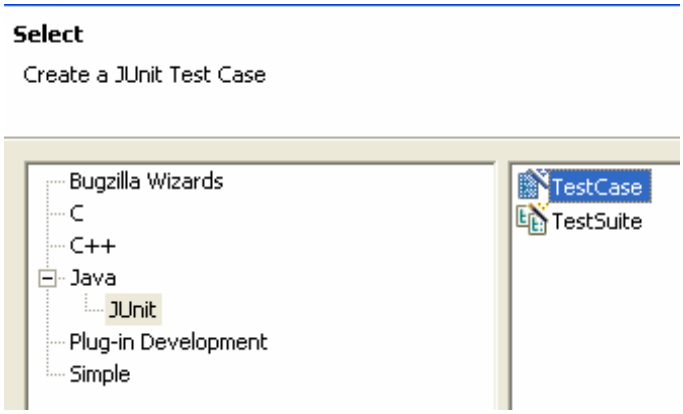

Figure - 2 On sélectionne « TestCase » (=scénario de test)

- Sous l'assistant :
  - cliquer sur « parcourir » pour le champ « Test Class » pour sélectionner la classe à tester.
  - dans la nouvelle fenêtre entrer le nom de la classe dans la zone de texte, le nom du scénario de test est automatiquement rempli en « MaClasseTest ».
  - des options représentées par des checkboxes sont sélectionnables :

    « *public static void main(String[] args)* » est utilisée pour lancer chaque test individuellement, comme si ils étaient une application unique.

    « *setUp()* » est une fonction qui peut être appelée avant exécution du test : on y initialise les objets nécessaires au test.

    « *tearDown()* » Cette fonction est appelée après exécution du test : les objets crées dans setUp() y sont détruits.

    Ces deux dernières fonctions sont utilisées pour réduire les redondances de code.

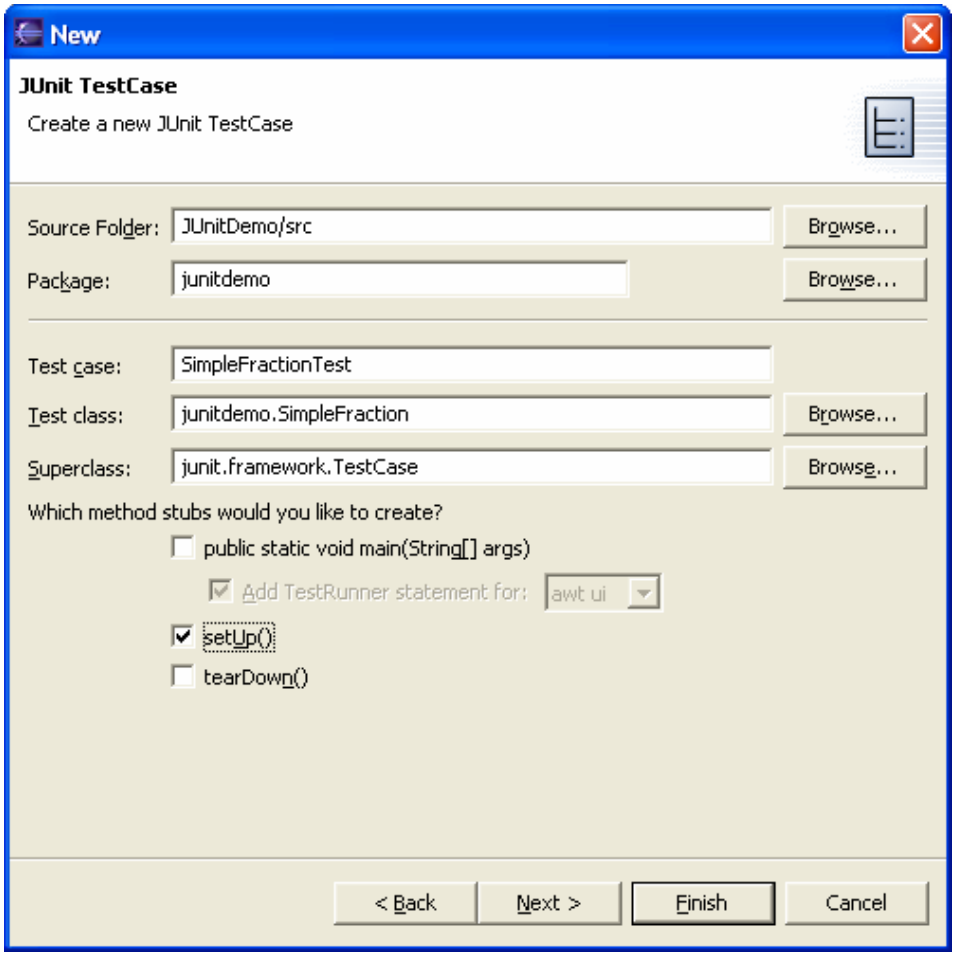

Figure - 3 On peut générer des méthodes de test supplémentaires en cochant les checkboxes

- on choisit ensuite sur la fenêtre suivante « New » quelles méthodes est ce que l'on désire tester ; le choix se fait par checkboxes. On clique enfin sur « finish » ce qui permet de générer la classe de test MaClasseTest avec les méthodes à tester. Il faut maintenant écrire les tests pour les méthodes sélectionnées.

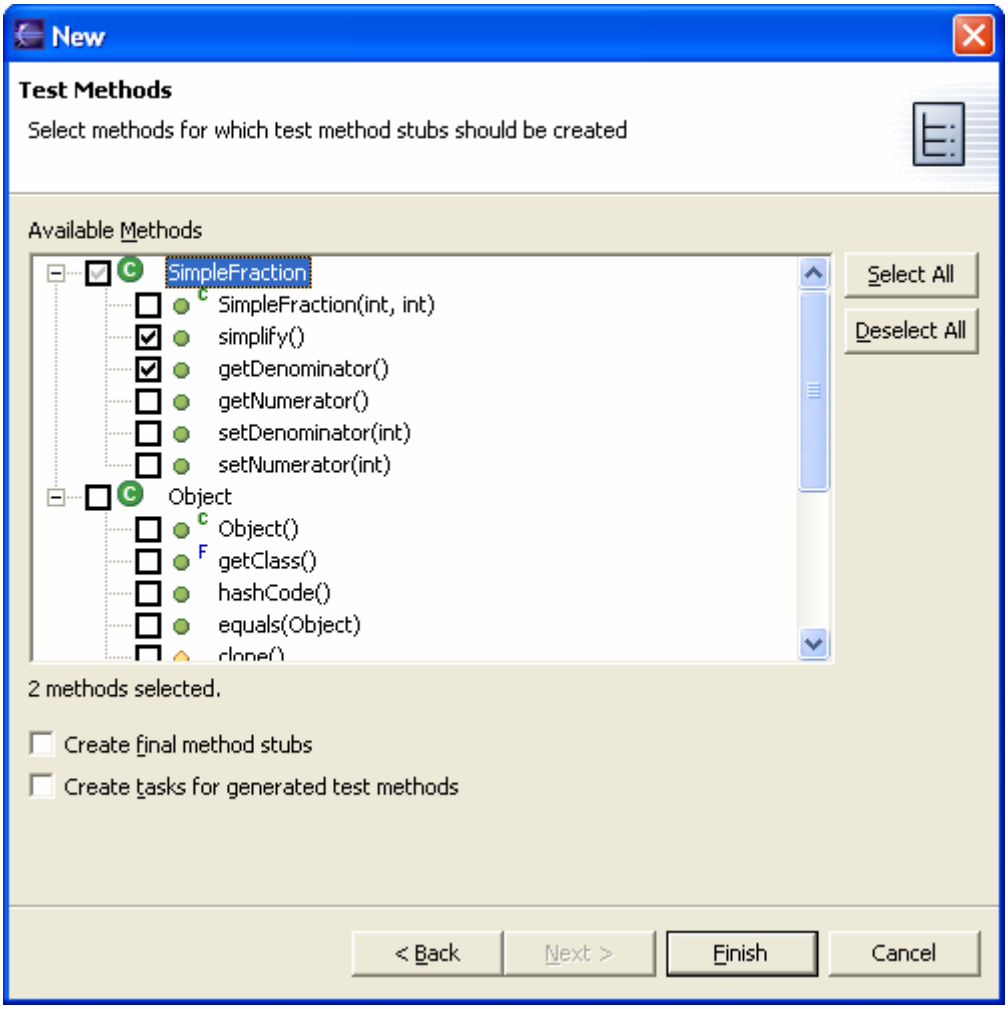

Figure - 4 Les checkboxes permettent de spécifier quelles méthodes de la classe sont à tester

1.4 ECRIRE LES TESTS

Les classes de test du framework JUNIT utilisent des méthodes JAVA dites d'*affirmation* (en anglais, « *assertion* »). On compte parmi celles - ci les méthodes :

- AssertEquals(int a, int b)
- AssertTrue(bool condition) et assertFalse
- assertSame et assertNotSame
- assertNull et assertNotNull
- fail()

Une description des méthodes existantes pour le framework JUNIT est donnée à l'adresse http://www.lcc.uma.es/docencia/ETSIInf/isd/JUnit/junit.3.8.1/javadoc/junit/framework/

1.5 LANCER LES TESTS

Pour exécuter un test JUNIT aller dans le menu ***Exécuter*** puis ***Exécuter…***

Dans l'arborescence sélectionner JUNIT avec choisir « ***New*** » après click avec le bouton droit de la souris. La classe de test concernée est alors automatiquement générée dans la fenêtre. Pour exécuter la classe de test il suffit de cliquer sur le bouton « ***Run*** ».

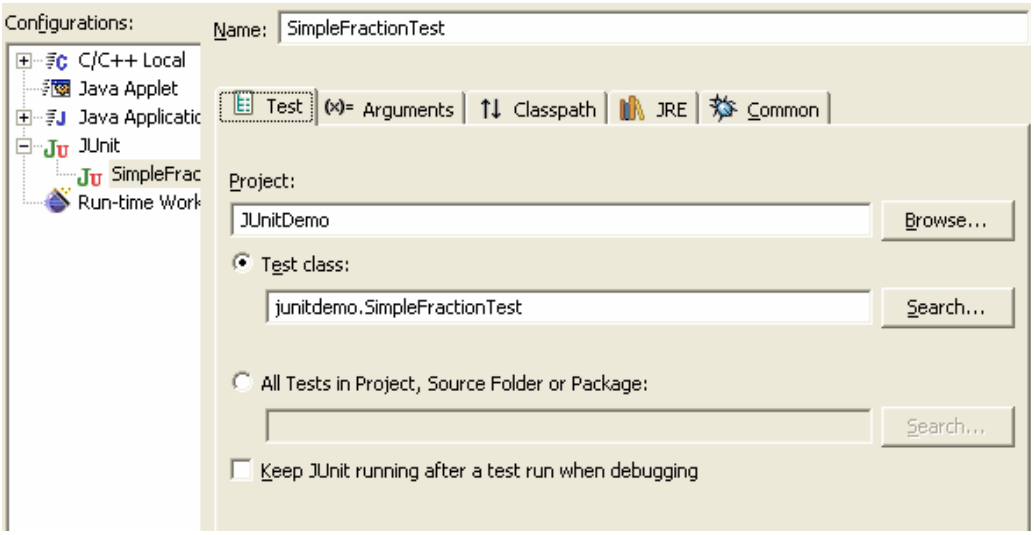

Figure - 5 Le menu suivant permet d'exécuter le test sur la classe à tester

1.6 INTERPRETATION DES RESULTATS

Si le test est lancé avec succès, cela génère une barre verte indiquant que le test s'est correctement exécuté.
JUnit test run finished : 0 seconds (Errors : 0, Failures : 0)

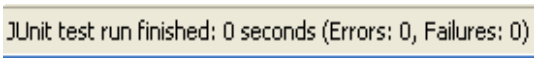

Figure - 6 Le test a réussi

Si le test échoue cela affiche une barre rouge avec un « Failure Trace » et une possibilité de déboguer le test qui vient d'être exécuté.
Junit.framework.AssertionFailedError etc.

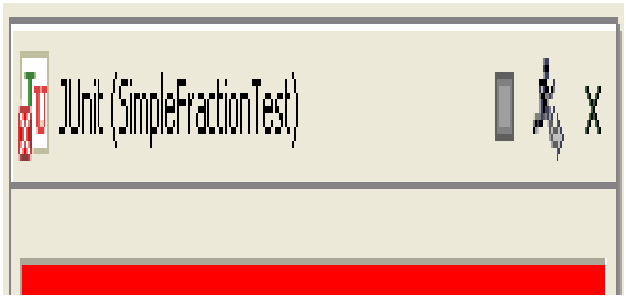

Figure - 7 Le test a échoué

2. MISE EN ŒUVRE SUR UNE CLASSE EN JAVA

Application à un exemple concret : La classe **DateUtil.java** du projet GIMA WEB

Cette classe se situe dans : *itrec -> framework -> common -> utils -> html*

Cette classe se compose de plusieurs méthodes qui effectuent des conversions de dates, heures, récupération de la date, de l'heure courante, etc.

Dans les options de Junit on génére des méthodes de test pour toutes les méthodes existantes de la classe DateUtil.java.

La classe **DateUtil.java** se compose du code suivant :

```
package com.itrec.framework.common.util;

import java.sql.Timestamp;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.Calendar;
import java.util.Date;
import java.util.GregorianCalendar;

/**
 * Objet technique permettant la manipulation de dates.
 * @author FMART
 * @version 2.00
 * <p>Crée le : 28 janv. 2004 par FMART (version 2.00)
 * <p>Modifiée le :     xx xxxx xxxx par xxxxx (version 2.xx)
 * <p>Copyright Adelior©2001
 */
public class DateUtil extends BaseUtil  {

      SimpleDateFormat dateFormat= new SimpleDateFormat("dd/MM/yyyy");
      SimpleDateFormat timeFormat= new SimpleDateFormat("hh:mm:ss");

      /**
       * Convertie une chaine date en Timestamp.
       * @param sdate Chaine date
       * @return Timestamp.
       * @throws ParseException Si une erreur survient.
       * <p>Crée le : 28 janv. 2004 par FMART (version 2.00)
       */
      public Timestamp convertToTimestamp(String sdate) throws
      ParseException {
            if ((sdate!=null)&&(sdate.length()>0)) {
            String newdate =
            this.formatStringDateTo(sdate.substring(0,10), "yyyy-MM-
            dd");
            if (sdate.length()>10) {
                  return
                  Timestamp.valueOf(newdate+sdate.substring(10));
                  } else {
                        return Timestamp.valueOf(newdate+" 00:00:00");
                  }
            } else return null;
      }
```

```
/**
 * Convertie un Timestamp en chaine au format JJ/MM/AAAA.
 * @param timestamp Timestamp
 * @return Date et Heure.
 * @throws ParseException Si une erreur survient.
 * <p>Exemple :
 * <p> Timestamp t = Timestamp.valueOf("2004-01-01 00:00:00");
 * <p> toString(t) retourne "01/01/2004"
 * <p>Crée le : 10 juin 2004 par FMART (version 2.00)
 * <p>Modifiée le :     xx xxxx xxxx par xxxxx (version 2.xx)
 */
public String toString(Timestamp timestamp) throws
ParseException {
      if (timestamp!=null) {
            Date date = timestamp;
            return dateFormat.format(date);
      } else return null;
}


/**
 * Convertie un Timestamp en chaine respectant le format
 * spécifié.
 * @param timestamp Timestamp
 * @param format Chaine de formatage
 * @return Date et Heure
 * @throws ParseException Si une erreur survient.
 * <p>Crée le : 28 janv. 2004 par FMART (version 2.00)
 * <p>Modifiée le :     xx xxxx xxxx par xxxxx (version 2.xx)
 */
public String toString(Timestamp timestamp, String format)
throws ParseException {
      if ((timestamp!=null)&&(format!=null)&&(format.length()>0))
{
            Date date = timestamp;
            SimpleDateFormat currentdateFormat = new
            SimpleDateFormat(format);
            return currentdateFormat.format(date);
      } else return null;
}


/**
 * Convertie une Date en chaine au format JJ/MM/AAAA.
 * @param date Date
 * @return Chaine date.
 * @throws ParseException Si une erreur survient.
 * <p>Crée le : 28 janv. 2004 par FMART (version 2.00)
 * <p>Modifiée le :     xx xxxx xxxx par xxxxx (version 2.xx)
 */
public String toString(Date date) throws ParseException {
      if (date!=null) {
            return dateFormat.format(date);
      } else return null;
}
```

```
/**
 * Convertie une Date en chaine respectant le format spécifié.
 * @param date Date
 * @param format Chaine de formatage
 * @return Chaine date.
 * @throws ParseException Si une erreur survient.
 * <p>Crée le : 28 janv. 2004 par FMART (version 2.00)
 * <p>Modifiée le :     xx xxxx xxxx par xxxxx (version 2.xx)
 */
public String toString(Date date, String format) throws
ParseException {
      if ((date!=null)&&(format!=null)&&(format.length()>0)) {
            SimpleDateFormat currentdateFormat = new
            SimpleDateFormat(format);
            return currentdateFormat.format(date);
      } else return null;
}


/**
 * Convertie un Timestamp en chaine au format JJ/MM/AAAA
 * HH:MM:SS.
 * @param timestamp Timestamp
 * @return Date et Heure.
 * @throws ParseException Si une erreur survient.
 * <p>Crée le : 28 janv. 2004 par FMART (version 2.00)
 * <p>Modifiée le :     xx xxxx xxxx par xxxxx (version 2.xx)
 */
public String getTime(Timestamp timestamp) throws ParseException
{
      if (timestamp!=null) {
            String time ="";
            GregorianCalendar calendar = new GregorianCalendar();
            Date date = timestamp;
            calendar.setTime(date);
            int value = calendar.get(Calendar.HOUR_OF_DAY);
            if (value<10) time=time+"0"+value; else
            time=time+value;
            value = calendar.get(Calendar.MINUTE);
            if (value<10) time=time+":0"+value; else
            time=time+":"+value;
            value = calendar.get(Calendar.SECOND);
            if (value<10) time=time+":0"+value; else
            time=time+":"+value;
            return time;
      } else return null;
}
```

```java
/**
 * Retourne la date courante sous forme de chaine au format
 * JJ/MM/AAAA.
 * @return Date courante.
 * <p>Crée le : 28 janv. 2004 par FMART (version 2.00)
 * <p>Modifiée le :     xx xxxx xxxx par xxxxx (version 2.xx)
 */
public String getCurrentDate() {
      Date currentDate = new Date(System.currentTimeMillis());
      return dateFormat.format(currentDate);
}


/**
 * Retourne l'heure courante sous forme de chaine au format
 * HH:MM:SS.
 * @return Heure courante.
 * <p>Crée le : 28 janv. 2004 par FMART (version 2.00)
 * <p>Modifiée le :     xx xxxx xxxx par xxxxx (version 2.xx)
 */
public String getCurrentTime() {
      String time ="";
      GregorianCalendar calendar = new GregorianCalendar();
      int value = calendar.get(Calendar.HOUR_OF_DAY);
      if (value<10) time=time+"0"+value; else time=time+value;
      value = calendar.get(Calendar.MINUTE);
      if (value<10) time=time+":0"+value; else
      time=time+":"+value;
      value = calendar.get(Calendar.SECOND);
      if (value<10) time=time+":0"+value; else
      time=time+":"+value;
      return time;
}


/**
 * Ajoute un nombre de jours à une date.
 * @param date Date
 * @param days Nombre de jour
 * @return Nouvelle date.
 * @throws ParseException Si une erreur survient.
 * <p>Crée le : 28 janv. 2004 par FMART (version 2.00)
 * <p>Modifiée le :     xx xxxx xxxx par xxxxx (version 2.xx)
 */
public Date add(Date date, int days) throws ParseException {
      if (date!=null) {
            GregorianCalendar calendar = this.getCalendar(date);
            calendar.add(Calendar.DATE, days);
            return calendar.getTime();
      } else return null;
}
```

```
/**
 * Ajoute un nombre de jours à une chaine date au format
 * JJ/MM/AAAA.
 * @param sdate Chaine date
 * @param days Nombre de jours
 * @return Nouvelle chaine date.
 * @throws ParseException Si une erreur survient.
 * <p>Crée le : 28 janv. 2004 par FMART (version 2.00)
 * <p>Modifiée le :     xx xxxx xxxx par xxxxx (version 2.xx)
 */
public String add(String sdate, int days) throws ParseException
{
      GregorianCalendar calendar = this.getCalendar(sdate);
      calendar.add(Calendar.DATE, days);
      return dateFormat.format(calendar.getTime());
}


/**
 * Ajoute un nombre de jours à une chaine date en respectant le
 * formatage spécifié.
 * @param sdate Chaine date
 * @param format Chaine de formatage
 * @param days Nombre de jours
 * @return Nouvelle chaine date.
 * @throws ParseException Si une erreur survient.
 * <p>Crée le : 28 janv. 2004 par FMART (version 2.00)
 * <p>Modifiée le :     xx xxxx xxxx par xxxxx (version 2.xx)
 */
public String add(String sdate, String format, int days) throws
ParseException {
      if((sdate!=null)&&(sdate.length()>0)&&(format!=null)&&(form
      at.length()>0)) {
            GregorianCalendar calendar = this.getCalendar(sdate,
            format);
            calendar.add(Calendar.DATE, days);
            return dateFormat.format(calendar.getTime());
      } else return null;
}


/**
 * Retourne le nombre de jours entre deux Dates.
 * @param dateDebut Date de début
 * @param dateFin Date de fin
 * @return Nombre de jours.
 * <p>Crée le : 28 avr. 2004 par FMART (version 2.00)
 * <p>Modifiée le :     xx xxxx xxxx par xxxxx (version 2.xx)
 */
public int getDayOfPeriod(Date dateDebut, Date dateFin) {
      // Retourne le nombre de jours
      return (int) ((dateFin.getTime()-
      dateDebut.getTime())/(long)86400000);
}
```

```
/**
 * Retourne le nombre de jours entre deux chaines date.
 * @param sdateDebut Chaine date de début
 * @param sdateFin Chaine date de fin
 * @return Nombre de jours.
 * @throws ParseException Si une erreur survient.
 * <p>Crée le : 28 avr. 2004 par FMART (version 2.00)
 * <p>Modifiée le :     xx xxxx xxxx par xxxxx (version 2.xx)
 */
public int getDayOfPeriod(String sdateDebut, String sdateFin)
throws ParseException {
      if((sdateDebut!=null)&&(sdateFin!=null)&&(sdateDebut.length
      ()>0
      )&&(sdateFin.length()>0)) {
            Date date1 = dateFormat.parse(sdateDebut);
            Date date2 = dateFormat.parse(sdateFin);
            return this.getDayOfPeriod(date1, date2);
      }else return 0;
}


/**
 * Convertie une Date en GregorianCalendar.
 * @param date Date à convertir
 * @return GregorianCalendar.
 * @throws ParseException Si une erreur survient.
 * <p>Crée le : 28 janv. 2004 par FMART (version 2.00)
 * <p>Modifiée le :     xx xxxx xxxx par xxxxx (version 2.xx)
 */
public GregorianCalendar getCalendar(Date date) throws
ParseException
{
      if (date!=null) {
            GregorianCalendar calendar = new GregorianCalendar();
            calendar.setTime(date);
            return calendar;
      } else return null;
}


/**
 * Convertie une chaine date au format JJ/MM/AAAA en
 * GregorianCalendar.
 * @param sdate Chaine date
 * @return GregorianCalendar.
 * @throws ParseException Si une erreur survient.
 * <p>Crée le : 28 janv. 2004 par FMART (version 2.00)
 */
public GregorianCalendar getCalendar(String sdate) throws
ParseException {
      if ((sdate!=null)&&(sdate.length()>0)) {
            Date date = this.toDate(sdate);
            GregorianCalendar calendar = new GregorianCalendar();
            calendar.setTime(date);
            return calendar;
      } else return null;
}
```

```
/**
 * Convertie une chaine date en GregorianCalendar.
 * @param sdate Chaine date
 * @param format Chaine de formatage
 * @return GregorianCalendar.
 * @throws ParseException Si une erreur survient.
 * <p>Crée le : 28 janv. 2004 par FMART (version 2.00)
 * <p>Modifiée le :     xx xxxx xxxx par xxxxx (version 2.xx)
 */
public GregorianCalendar getCalendar(String sdate, String
format) throws ParseException {
      if((sdate!=null)&&(sdate.length()>0)&&(format!=null)&&(form
      at.length()>0)) {
            Date date = this.toDate(sdate, format);
            GregorianCalendar calendar = new GregorianCalendar();
            calendar.setTime(date);
            return calendar;
      } else return null;
}


/**
 * Convertie une chaine date en une Date au format JJ/MM/AAAA.
 * @param sdate Chaine date
 * @return Date formatée.
 * @throws ParseException Si une erreur survient.
 * <p>Crée le : 28 janv. 2004 par FMART (version 2.00)
 * <p>Modifiée le :     xx xxxx xxxx par xxxxx (version 2.xx)
 */
public Date toDate(String sdate) throws ParseException{
      if ((sdate!=null)&&(sdate.length()>0)) {
            return dateFormat.parse(sdate);
      } else return null;
}


/**
 * Convertie une chaine date en une Date formatée.
 * @param sdate Chaine date
 * @param format Chaine de formatage
 * @return Date formatée.
 * @throws ParseException Si une erreur survient.
 * <p>Crée le : 28 janv. 2004 par FMART (version 2.00)
 * <p>Modifiée le :     xx xxxx xxxx par xxxxx (version 2.xx)
 */
public Date toDate(String sdate, String format) throws
ParseException{
      if((sdate!=null)&&(sdate.length()>0)&&(format!=null)&&(form
      at.length()>0)) {
            SimpleDateFormat currentdateFormat = new
            SimpleDateFormat(format);
            return currentdateFormat.parse(sdate);
      } else return null;
}
```

```
/**
 * Formate une chaine date à partir d'une chaine de formatage.
 * @param sdate Chaine date
 * @param format Chaine de formatage
 * @return Chaine date formatée.
 * @throws ParseException Si une erreur survient.
 * <p>Crée le : 28 janv. 2004 par FMART (version 2.00)
 * <p>Modifiée le :     xx xxxx xxxx par xxxxx (version 2.xx)
 */
public String formatStringDateTo(String sdate, String format)
throws ParseException {
      if((sdate!=null)&&(sdate.length()>0)&&(format!=null)&&(form
      at.length()>0)) {
            SimpleDateFormat currentdateFormat = new
            SimpleDateFormat(format);
            return
            currentdateFormat.format(dateFormat.parse(sdate));
      } else return null;
}


/**
 * Retourne l'année courante.
 * @return Année courante.
 * <p>Crée le : 28 janv. 2004 par FMART (version 2.00)
 * <p>Modifiée le :     xx xxxx xxxx par xxxxx (version 2.xx)
 */
public String getCurrentYear() {
      Date currentDate = new Date(System.currentTimeMillis());
      return dateFormat.format(currentDate).substring(6,10);

}


/**
 * Retourne le mois courant.
 * @return Mois courant.
 * <p>Crée le : 28 janv. 2004 par FMART (version 2.00)
 * <p>Modifiée le :     xx xxxx xxxx par xxxxx (version 2.xx)
 */
public String getCurrentMonth() {
      Date currentDate = new Date(System.currentTimeMillis());
      return dateFormat.format(currentDate).substring(3,5);
}


/**
 * Retourne le jour courant.
 * @return Jour courant.
 * <p>Crée le : 28 janv. 2004 par FMART (version 2.00)
 * <p>Modifiée le :     xx xxxx xxxx par xxxxx (version 2.xx)
 */
public String getCurrentDay() {
      Date currentDate = new Date(System.currentTimeMillis());
      return dateFormat.format(currentDate).substring(0,2);
}
```

```
	/**
	 * Retoune l'année d'une chaine date.
	 * @param sdate Chaine date
	 * @return Année de la date.
	 * <p>Crée le : 28 janv. 2004 par FMART (version 2.00)
	 * <p>Modifiée le :     xx xxxx xxxx par xxxxx (version 2.xx)
	 */
	public String getYear(String sdate) {
		if ((sdate!=null)&&(sdate.length()>0)) {
			return sdate.substring(6,10);
		} else return null;
	}


	/**
	 * Retoune le mois d'une chaine date.
	 * @param sdate Chaine date
	 * @return Mois de la date.
	 * <p>Crée le : 28 janv. 2004 par FMART (version 2.00)
	 * <p>Modifiée le :     xx xxxx xxxx par xxxxx (version 2.xx)
	 */
	public String getMonth(String sdate) {
		if ((sdate!=null)&&(sdate.length()>0)) {
			return sdate.substring(3,5);
		} else return null;
	}


	/**
	 * Retoune le jour d'une chaine date.
	 * @param sdate Chaine date
	 * @return Jour de la date.
	 * <p>Crée le : 28 janv. 2004 par FMART (version 2.00)
	 * <p>Modifiée le :     xx xxxx xxxx par xxxxx (version 2.xx)
	 */
	public String getDay(String sdate) {
		if ((sdate!=null)&&(sdate.length()>0)) {
			return sdate.substring(0,2);
		} else return null;
	}

}
```

Le lancement d'un scénario de test Junit sous Eclipse a généré automatiquement le fichier DateUtilTest.java suivant :

```
package com.itrec.framework.common.util.html;

import junit.framework.TestCase;

/**
 * Description
 * <p>Crée le : 20 juil. 2004 par jdahan (version 2.00)
 * <p>Modifiée le :     xx xxxx xxxx par xxxxx (version 2.xx)
 * @author jdahan
 * @version 2.00
 * <p>Copyright Adelior©2001
 * <p>All right reserved
 */
public class DateUtilTest extends TestCase {

	/**
	 * Constructor for DateUtilTest.
	 * @param arg0
	 */
	public DateUtilTest(String arg0) {
		super(arg0);
	}

	public void testConvertToTimestamp() {
	}

	/*
	 * Test pour String toString(Timestamp)
	 */
	public void testToStringTimestamp() {
	}

	/*
	 * Test pour String toString(Timestamp, String)
	 */
	public void testToStringTimestampString() {
	}

	/*
	 * Test pour String toString(Date)
	 */
	public void testToStringDate() {
	}

	/*
	 * Test pour String toString(Date, String)
	 */
	public void testToStringDateString() {
	}
```

```
public void testGetTime() {
}

public void testGetCurrentDate() {
}

public void testGetCurrentTime() {
}

/*
 * Test pour Date add(Date, int)
 */
public void testAddDateint() {
}

/*
 * Test pour String add(String, int)
 */
public void testAddStringint() {
}

/*
 * Test pour String add(String, String, int)
 */
public void testAddStringStringint() {
}

/*
 * Test pour int getDayOfPeriod(Date, Date)
 */
public void testGetDayOfPeriodDateDate() {
}

/*
 * Test pour int getDayOfPeriod(String, String)
 */
public void testGetDayOfPeriodStringString() {
}

/*
 * Test pour GregorianCalendar getCalendar(Date)
 */
public void testGetCalendarDate() {
}

/*
 * Test pour GregorianCalendar getCalendar(String)
 */
public void testGetCalendarString() {
}

/*
 * Test pour GregorianCalendar getCalendar(String, String)
 */
public void testGetCalendarStringString() {
}
```

```
        /*
         * Test pour Date toDate(String)
         */
        public void testToDateString() {
        }

        /*
         * Test pour Date toDate(String, String)
         */
        public void testToDateStringString() {
        }

        public void testFormatStringDateTo() {
        }

        public void testGetCurrentYear() {
        }

        public void testGetCurrentMonth() {
        }

        public void testGetCurrentDay() {
        }

        public void testGetYear() {
        }

        public void testGetMonth() {
        }

        public void testGetDay() {
        }

}
```

A partir de la classe de test qui vient d'être générée, j'ai écrit les méthodes de test correspondant aux méthodes que je souhaite tester :

```
package com.itrec.framework.common.util;

import junit.framework.TestCase;
import java.sql.Timestamp;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.GregorianCalendar;


/**
 * Description
 * <p>Crée le : 21 juil. 2004 par jdahan (version 2.00)
 * <p>Modifiée le : xx xxxx 2004 par jdahan (version 2.xx)
 * @author jdahan
 * @version 2.00
 * <p>Copyright Adelior©2001
 * <p>All right reserved
 */


public class DateUtilTest extends TestCase {

	DateUtil d1 = new DateUtil();
	DateUtil d2 = new DateUtil();

	SimpleDateFormat dateFormat= new SimpleDateFormat("dd/MM/yyyy");
	SimpleDateFormat timeFormat= new SimpleDateFormat("hh:mm:ss");

	/*
	 * le constructeur de la classe DateUtilTest
	 */
	public DateUtilTest(String arg0) {
		super(arg0);
	}

	/*
	 * on peut déclarer ici les objets utilisés dans la classe de
	 * test
	 * (non implémenté dans Eclipse)
	 */
	protected void setUp() throws Exception {
		super.setUp();
	}


	/*
	 * on détruit ici les objets utilisés dans la classe de test
	 * (non implémenté dans Eclipse)
	 */
	protected void tearDown() throws Exception {
		super.tearDown();
	}
```

```
/*
 * Teste les fonctions de conversion de date
 * de type String en TimeStamp, fonction convertToTimestamp()
 * et de TimeStamp en String, fonction toString()
 */
public void testConvertToTimestampEtConvertToString() throws
ParseException {
      String dateString = dateFormat.format(new
      Date(System.currentTimeMillis()));
      /* Appel de la fonction convertToTimestamp() */
      Timestamp dateStamp = d1.convertToTimestamp(dateString);
      /* Appel de la fonction toString() */
      String dateStringAComparer = d1.toString(dateStamp);
      assertEquals(dateString,dateStringAComparer);
}


/*
 * Teste la fonction GetTime()
 */
public void testGetTime() throws ParseException {
      /* renvoie la date courante en FR */
      String dateString = dateFormat.format(new
      Date(System.currentTimeMillis()));
      /* renvoie la date en US + heure initialisee a zero */
      Timestamp dateStamp = d1.convertToTimestamp(dateString);
      /* Appel de la fonction GetTime() */
      /* renvoie l'heure a zero */
      String tempsZero = d1.getTime(dateStamp);
      assertEquals("00:00:00",tempsZero);
}


/*
 * Teste les fonctions GetCurrentTime() et GetCurrentDate().
 */
public void testGetCurrentTimeAndGetCurrentDate() throws
ParseException {
      /* test de la méthode GetCurrentTime() */
      String recupTime = d1.getCurrentTime();
      /* je teste que la chaîne retournée par la fonction
      getCurrentTime()
       * est bien codée sur 8 caracteres */
      assertEquals(8, recupTime.length());
      /* je teste que les heures ne depassent pas 12, que les
      minutes et les secondes ne dépassent pas 60 */
      String heureCourante = recupTime.substring(0,2);
      String minuteCourante = recupTime.substring(3,5);
      String secondeCourante = recupTime.substring(6,8);
      assertTrue("GetCurrentTime() a renvoye une heure = " +
      Integer.parseInt(heureCourante) +
      " alors que le max est 24.",
      Integer.parseInt(heureCourante)<24);
      assertTrue("GetCurrentTime() a renvoye des minutes = " +
      Integer.parseInt(minuteCourante) +
      " alors que le max est 60.",
      Integer.parseInt(heureCourante)<60);
```

```
        assertTrue("GetCurrentTime() a renvoye des secondes = " +
        Integer.parseInt(secondeCourante) +
        " alors que le max est 60.",
        Integer.parseInt(heureCourante)<60);

        /* test de la méthode GetCurrentDate() */
        String recupDate = d1.getCurrentDate();
        /* je teste que la chaîne retournée par la fonction
        getCurrentDate()
         * est bien codée sur 10 caracteres */
        assertEquals(10, recupDate.length());
        /* je teste que les jours ne depassent pas 31 et que les
        mois ne depassent pas 12 */
        String jourCourant = recupDate.substring(0,2);
        String moisCourant = recupDate.substring(3,5);
        assertTrue("GetCurrentDate() a renvoye un jour = " +
        Integer.parseInt(jourCourant) +
        " alors que le max est 31.",
        Integer.parseInt(heureCourante)<=31);
        assertTrue("GetCurrentDate() a renvoye un mois = " +
        Integer.parseInt(moisCourant) +
        " alors que le max est 12.",
        Integer.parseInt(moisCourant)<=12);

        /* test sur l'exactitude de la value renvoyée par
        GetCurrentTime() et GetCurrentDate() */
        String recupDateHeure = recupDate + " " + recupTime + ".0";

        Timestamp t = d1.convertToTimestamp(recupDateHeure);
        long millisec = t.getTime();

        Date maDate = new Date();
        long millisecAComparer = maDate.getTime();

        assertTrue("La difference entre la date/heure renvoyées par
        les fonctions getCurrentTime() et getCurrentDate() " +
        "et la date/heure système doit être inférieure à 1000.",
        millisec-millisecAComparer<1000);
}


/*
 * Ajoute un nombre de jours à une date.
 * Test pour Date add(Date, int)
 */
public void testAddDateint() throws ParseException {

        /* je recupere le date du jour à laquelle je rajoute un
        certain nombre de jours */
        Date dateJour = new Date(System.currentTimeMillis());
        /* retourne le nombre de millisecondes depuis le 1er
        janvier 1970 */
        long maDateJour = dateJour.getTime();
        long nbJour = 60;
        long millisecDateJour = maDateJour + nbJour*24*60*60*1000;

        /* je teste la fonction add() qui prend en paramètre la
```


ITREC GESTION
59, rue de Billancourt BP 56
92105 Boulogne Billancourt
Tél. : +33 (0)1 46 03 60 58
Fax : +33 (0)1 46 03 61 81
Site : www.itrec.com

```
        date du jour + le nombre de jours à ajouter */
        long millisecDateJourAComparer = d1.add(dateJour,
        (int)nbJour).getTime();

        /* je vérifie que les deux dates sont égales */
        assertEquals(millisecDateJour, millisecDateJourAComparer);
}


/*
 * Test pour String add(String, int)
 * Ajoute un nombre de jours à une chaine date au format
 * JJ/MM/AAAA.
 */
public void testAddStringint() throws ParseException {

        /* je recupere le date du jour à laquelle je rajoute un
        certain nombre de jours */
        long nbJour = 80;
        Date dateComplete = new Date(System.currentTimeMillis());
        /* retourne la date au format JJ/MM/AAAA sans le temps */
        String maDateJour = dateFormat.format(dateComplete);
        /* retourne la date complète avec le temps initialisé à
        zero */
        Date dateTronquee = d1.toDate(maDateJour);
        /* retourne le nombre de millisecondes depuis le 1er
        janvier 1970 */
        long millisecDateJour = dateTronquee.getTime() +
        nbJour*24*60*60*1000;

        /* je teste la fonction add() qui prend en paramètre la
        date du jour en chaîne + 5 jours */
        long millisecDateJourAComparer=d1.toDate(d1.add(maDateJour,
        (int)nbJour)).getTime();

        /* je vérifie que les deux dates sont égales */
        assertEquals(millisecDateJour, millisecDateJourAComparer);
}


/*
 * Test pour String add(String, String, int)
 * Ajoute un nombre de jours à une chaine date en respectant le
 * formatage spécifié.
 */
public void testAddStringStringint() throws ParseException {

        long NbJour = 85;

        /* je recupere le date du jour à laquelle je rajoute un
        certain nombre de jours */
        String s = dateFormat.format(new
        Date(System.currentTimeMillis()));
        Date DateJourAComparer = d1.toDate(s);
        long tmpsMilliSecAComparer = DateJourAComparer.getTime() +
        NbJour*24*60*60*1000;
```

```
		/* je teste la fonction String add(String, String, int) */
		String DateAjoutee = d1.add(dateFormat.format(new
		Date(System.currentTimeMillis())),"dd/MM/yyyy",
		(int)NbJour);
		Date DateJourAjoutee = d1.toDate(DateAjoutee);
		long tmpsMilliSec = DateJourAjoutee.getTime();

		/* je vérifie que les deux dates sont égales */
		assertEquals(tmpsMilliSec, tmpsMilliSecAComparer);
	}


	/*
	 * Test pour int getDayOfPeriod(Date, Date)
	 * Retourne le nombre de jours entre deux Dates.
	 */
	public void testGetDayOfPeriodDateDate() throws ParseException {
		long nbJours = 31;
		Date dateDepart = new Date();
		dateDepart.setDate(29);
		dateDepart.setMonth(06);
		dateDepart.setYear(2004);
		/* retourne la date au format JJ/MM/AAAA sans le temps */

		String maDateJour = dateFormat.format(dateDepart);
		/* retourne la date de début (DD) complète avec le temps
		initialisé à zero */
		Date DD = d1.toDate(maDateJour);
		/* retourne la date de fin (DF) complète avec nbJours
		ajouté et le temps initialisé à zero */
		Date DF = d1.add(DD,(int)nbJours);
		/* je teste la fonction getDayOfPeriod() */
		int nbJoursAComparer = d1.getDayOfPeriod(DD, DF);
		/* je vérifie que les deux "nombre de jours" sont égaux */
		assertEquals(nbJours,nbJoursAComparer);
	}


	/*
	 * Test pour int getDayOfPeriod(String, String)
	 * Retourne le nombre de jours entre deux chaines de dates.
	 */
	public void testGetDayOfPeriodStringString() throws
	ParseException {
		long nbJours = 81;
		Date dateDepart = new Date();
		dateDepart.setDate(29);
		dateDepart.setMonth(06);
		dateDepart.setYear(2004);
		/* retourne la date au format JJ/MM/AAAA sans le temps */
		String maDateJour = dateFormat.format(dateDepart);
		Date DD = d1.toDate(maDateJour);
		/* retourne la date de fin (DF) complète avec nbJours
		ajouté et le temps initialisé à zero */
		Date DF = d1.add(DD,(int)nbJours);
		/* je teste la fonction getDayOfPeriod() */
		int nbJoursAComparer = d1.getDayOfPeriod(d1.toString(DD),
```


ITREC GESTION
59, rue de Billancourt BP 56
92105 Boulogne Billancourt
Tél. : +33 (0)1 46 03 60 58
Fax : +33 (0)1 46 03 61 81
Site : www.itrec.com

```
                d1.toString(DF));
                /* je vérifie que les deux "nombre de jours" sont égaux */
                assertEquals(nbJours,nbJoursAComparer);
        }


        /*
         * Test pour GregorianCalendar getCalendar(Date)
         * Convertie une Date en GregorianCalendar.
         */
        public void testGetCalendarDate() throws ParseException {
                Date maDate = new Date();
                long millisec = maDate.getTime();
                /* je teste la fonction getCalendar() */
                GregorianCalendar monCalendar = d1.getCalendar(new Date());
                /* je vérifie que les valeurs des variables sont égales */
                assertEquals(millisec,monCalendar.getTime().getTime());
        }


        /*
         * Test pour GregorianCalendar getCalendar(String)
         * Convertie une chaine date au format JJ/MM/AAAA en
         * GregorianCalendar.
         */
        public void testGetCalendarString() throws ParseException {
                Date dateComplete = new Date(System.currentTimeMillis());
                /* retourne la date au format JJ/MM/AAAA sans le temps */
                String maDateJour = dateFormat.format(dateComplete);
                /* retourne la date complète avec le temps initialisé à
                zero */
                Date dateTronquee = d1.toDate(maDateJour);
                long millisec = dateTronquee.getTime();
                /* je teste la fonction getCalendar() */
                GregorianCalendar monCalendar =
                d1.getCalendar(d1.toString(new Date()));
                /* je vérifie que les valeurs des variables sont égales */
                assertEquals(millisec,monCalendar.getTime().getTime());
        }


        /*
         * Test pour GregorianCalendar getCalendar(String, String)
         * Convertie une chaine date en GregorianCalendar.
         * @param sdate Chaine date
         * @param format Chaine de formatage
         */
        public void testGetCalendarStringString() throws ParseException
{
                Date dateComplete = new Date(System.currentTimeMillis());
                /* retourne la date au format JJ/MM/AAAA sans le temps */
                String maDateJour = dateFormat.format(dateComplete);
                /* retourne la date complète avec le temps initialisé à
                zero */
                Date dateTronquee = d1.toDate(maDateJour);
                long millisec = dateTronquee.getTime();
                /* je teste la fonction getCalendar() */
```

```
        GregorianCalendar monCalendar =
        d1.getCalendar(d1.toString(new
        Date()),"dd/MM/yyy");
        /* je vérifie que les valeurs des variables sont égales */
        assertEquals(millisec,monCalendar.getTime().getTime());
}


/*
 * Test pour Date toDate(String)
 * Convertie une chaine date en une Date au format JJ/MM/AAAA.
 */
public void testToDateString() throws ParseException{
      /* retourne la date au format JJ/MM/AAAA, heure initialisee
      a zero */
      String maDateJour = dateFormat.format(new
      Date(System.currentTimeMillis()));
      /* je teste la fonction toDate(String) */
      Date dateATester = d1.toDate(maDateJour);
      long dateATesterEnMillisecondes = dateATester.getTime();
      /* renvoie la date en US, heure initialisee a zero */
      Timestamp dateStamp = d1.convertToTimestamp(maDateJour);
      long dateAComparerEnMillisecondes = dateStamp.getTime();
      /* je vérifie que les valeurs des variables sont égales */

      assertEquals(dateATesterEnMillisecondes,
      dateAComparerEnMillisecondes);
}


/*
 * Test pour Date toDate(String, String)
 * Convertie une chaine date en une Date formatée.
 */
public void testToDateStringString() throws ParseException {
      /* retourne la date au format JJ/MM/AAAA, heure initialisee
      a zero */
      String maDateJour = dateFormat.format(new
      Date(System.currentTimeMillis()));

      /* je teste la fonction toDate(String) */
      Date dateATester = d1.toDate(maDateJour,"dd/MM/yyy");
      long dateATesterEnMillisecondes = dateATester.getTime();

      Timestamp dateStamp = d1.convertToTimestamp(maDateJour);
      /* renvoie la date en US, heure initialisee a zero */
      long dateAComparerEnMillisecondes = dateStamp.getTime();
      /* je vérifie que les valeurs des variables sont égales */

      assertEquals(dateATesterEnMillisecondes,
      dateAComparerEnMillisecondes);
}
```

```
	/*
	 * Test pour String formatStringDateTo(String sdate, String
	 * format)
	 * Formate une chaine date à partir d'une chaine de formatage.
	 */
	public void testFormatStringDateTo() throws ParseException {
		/* retourne la date au format JJ/MM/AAAA, heure initialisee
		a zero */
		String maDateJour = dateFormat.format(new
		Date(System.currentTimeMillis()));
		/* je teste la fonction String formatStringDateTo(String
		sdate, String format) */
		String dateATester =
		d1.formatStringDateTo(maDateJour,"dd/MM/yyyy");

		/* je vérifie que les valeurs des variables sont égales */

		assertEquals(dateATester, maDateJour);
	}

	/*
	 * Test pour String getCurrentDay()
	 * Retourne le jour courant au format chaîne.
	 */
	public void testGetCurrentDay() {
		String testDay = d1.getCurrentDay();
		String recupDate = d1.getCurrentDate();
		String jourCourant = recupDate.substring(0,2);
		assertEquals(jourCourant, testDay);
	}


	/*
	 * Test pour String getCurrentMonth()
	 * Retourne le mois courant au format chaîne.
	 */
	public void testGetCurrentMonth() {
		String testMonth = d1.getCurrentMonth();
		String recupDate = d1.getCurrentDate();
		String moisCourant = recupDate.substring(3,5);
		assertEquals(moisCourant, testMonth);
	}


	/*
	 * Test pour String getCurrentYear()
	 * Retourne l'année courante au format chaîne.
	 */
	public void testGetCurrentYear() {
		String testAnnee = d1.getCurrentYear();
		String recupDate = d1.getCurrentDate();
		String anneeCourante = recupDate.substring(6,10);

		assertEquals(anneeCourante, testAnnee);
	}
}
```

CONCLUSION

Ce document technique offre les informations nécessaires permettant à un non initié d'intégrer le framework Junit dans la plate-forme de développement Eclipse afin d'écrire des tests unitaires sur des méthodes écrites en langage Java.

Le test en lui même doit pouvoir servir de documentation technique : il apporte de la valeur documentaire au code. Le test permet également de s'affranchir d'un autre type de problème : ne pas être obligé de coder l'interface graphique tant que le métier n'est pas implémenté, et ne pas dépendre du débogage de l'IHM elle-même pendant le codage du métier.

Avant de livrer une application, il est important de prendre le temps de la tester de manière exhaustive; en effet, plus on teste régulièrement des parties d'application, plus le bug est facilement détectable. Il en est de même lorsque des modifications sont apportées à l'application. On peut ainsi passer des heures à déboguer un petit rien alors que quelques secondes suffisent à un test pour mettre toute une application à l'épreuve. Enfin, la non régression du code est respectée : on peut modifier un code pour qu'il résiste au pire et l'on sait immédiatement si il y a un bug, où et pourquoi.. cela permet donc au développeur d'avancer plus vite dans son travail.