מעבדה: בדיקות תקינות ושימוש ב-Junit.

רקע:

בהרצאות עברנו על חשיבות איכות הקוד, והצורך בבדיקתו. במעבדה זו נעבור על חלק הבדיקות המתבצע תוך כדי כתיבת הקוד על ידי המתכנת תוך שימוש ברכיב ה Junit המשמש למטרה זו.

דוגמא מלווה:

במעבדה זו נעשה שימוש במערכת לחישוב פיננסי שעושה שימוש במטבעות שונים. הוספה והחסרה של סכום באותו מטבע ($ לדוגמא) היא קלה למדי. הקושי (או הענין – תלוי איך מסתכלים) מתחיל כשהפעולה נוגעת למטבעות שונים (לדוגמא 20$ - 40€), והתוצאה תלויה בשערי החליפין שנגזרים ממועד הפעולה.

נתחיל בהגדרת מחלקת כסף Money, ששומרת את סכום הכסף כמספר, ואת המטבע ע"י ייצוג של שלוש אותיות לפי הסטנדרט המקובל (USD, NIS, GBP וכו'). קוד המחלקה מופיעה באתר.

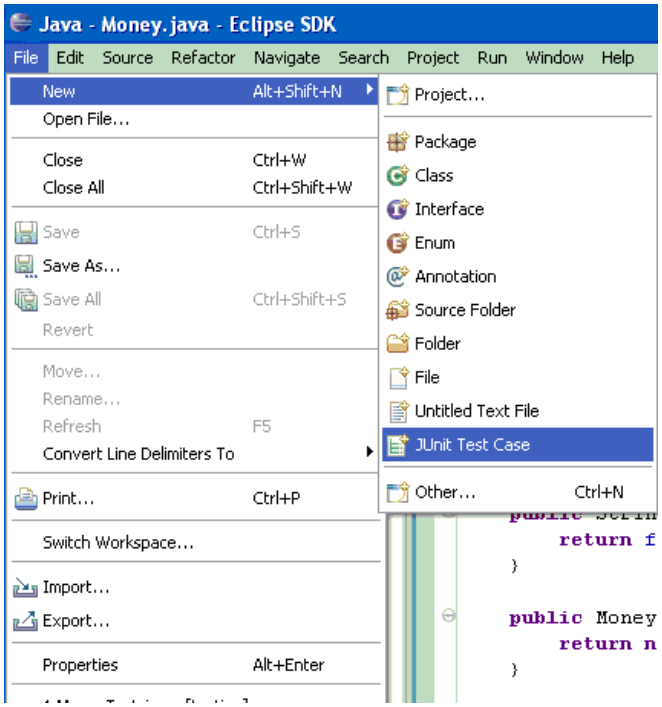
ממש את המתודה Add שמופעלת כשרוצים לסכום שני סכומי כסף (מאותו מטבע).
מתודה זו מחזירה אוביקט חדש מסוג Money(עם הערכים המתאימים כמובן).

לפני שנמשיך הלאה, בואו נבדוק את הקוד שכתבתם. לצורך כך נכתוב תוכנית בדיקה. כל תוכנית בדיקה שעושה שימוש ב Junit מרחיבה את המחלקה TestCase.

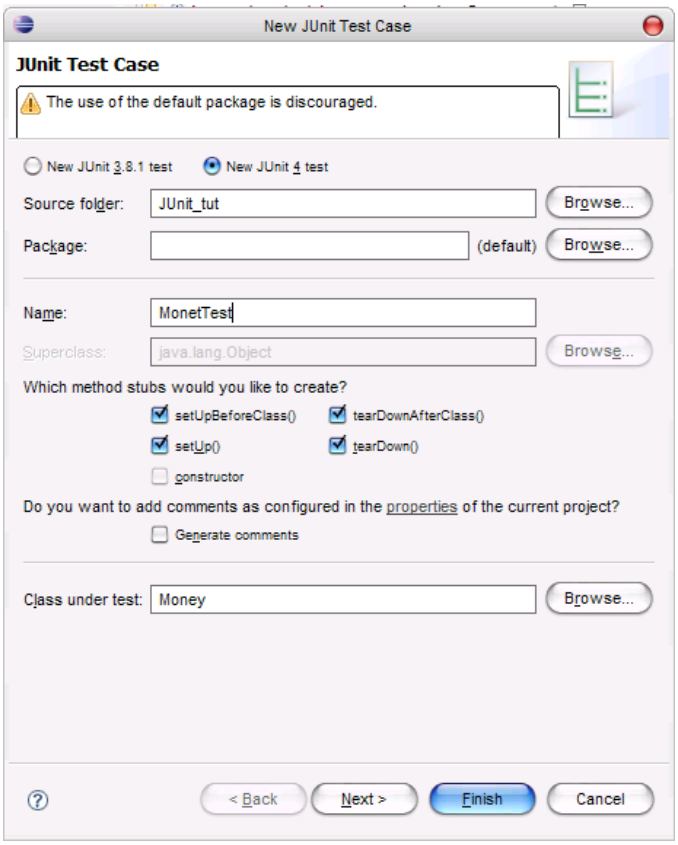
יצרו TestCase חדש ע"י File->new -> Junit..

יפתח חלון בו תדרשו לתת שם למחלקה ולציין את המחלקה אותה אתם בודקים.

שימו לב שאנו שמים את המחלקה באותה חבילה של המחלקה הנבדקת כדי לאפשר גישה למתודות השמורות.



אשרו את החלון שנפתח, וקבלתם מחלקת בדיקות חדשה.



בשלב הבא נכניס תכנים למחלקה זו.
כל בדיקה במחלקה תמומש ע"י מתודה.
כל מתודה תורכב משלושה חלקים:
1 יצירת האוביקטים שנעזר בהם לבדיקה
2 ביצוע הפעולה שאת תוצאותיה נבדוק
3 וידוא תקינות הפעולה (קרי השוואת התוצאות לצפי).

```
public class MoneyTest extends TestCase {
    //…
    public void testSimpleAdd() {
        Money m12CHF= new Money(12, "CHF");   // (1)
        Money m14CHF= new Money(14, "CHF");
        Money expected= new Money(26, "CHF");
        Money result= m12CHF.add(m14CHF);     // (2)
        Assert.assertTrue(expected.equals(result));      // (3)
    }
}
```

הוסיפו את התוכן הנ"ל למתודה testSimpleAdd().

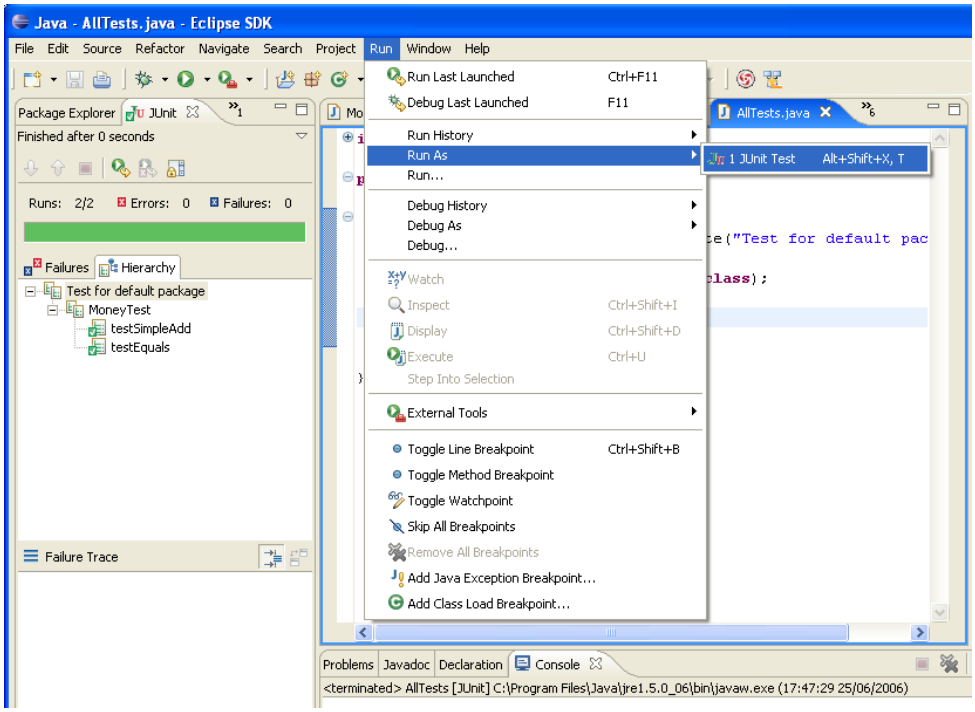
כל הכרזת Assert נבדקת, ואם AssertTrue לא מתקיים, נשלחת הודעת שגיאה ל JUnit.
AssertEquals משווה ומעבר לבדיקה, מדפיס את ערכי האובייקטים המושווים, במידה ונמצא הבדל ביניהם.

כדי להשוות בין התוצאות עלינו להשוות בין שני אובייקטים של Money. כדי שנוכל להשוות ביניהם עלינו לדרוס את המתודה equals ששייכת לObject. שני אובייקטים יהיו שווים אם יכילו את אותו סכום ואותו מטבע.

הוסף את הקוד הבא למחלקה Money

```
public boolean equals(Object anObject) {
    if (anObject instanceof Money) {
        Money aMoney= (Money)anObject;
        return aMoney.currency().equals(currency())
            && amount() == aMoney.amount();
    }
    return false;
}
```

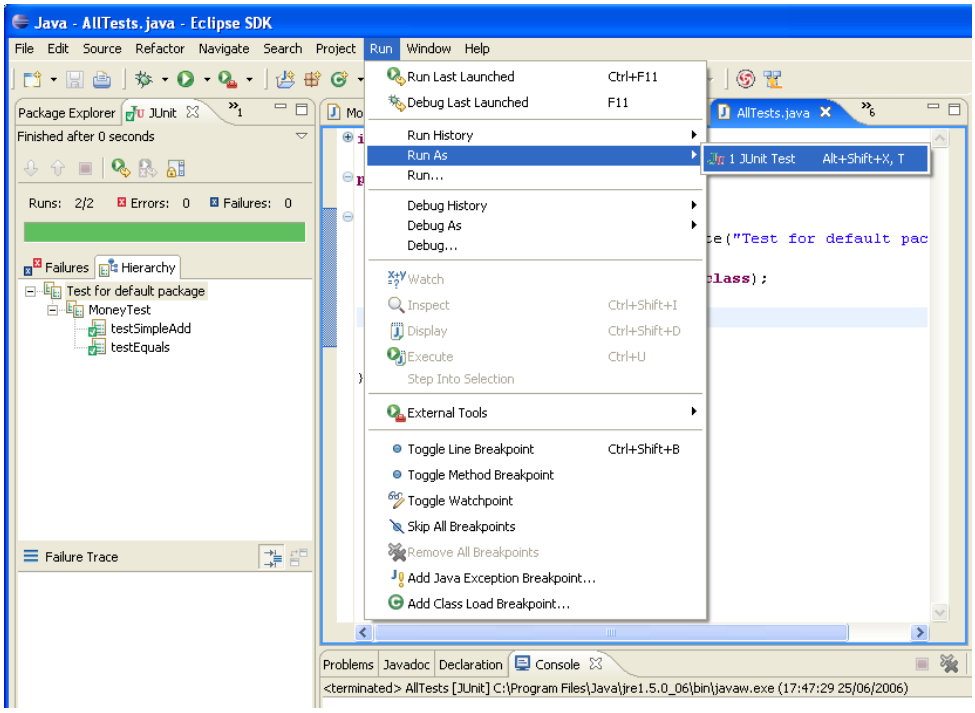
כעת נוכל להשוות בין אובייקטים שונים של Money.
הריצו את מחלקת ה TestCase שיצרתם ותקבלו את החלון הבא: כאמור בדקנו כרגע רק את המתודה testSimpleAdd().
שימו לב שניתן להריץ את הטסט בתוך ה eclipse ע"י Run->Run as..->Junit Test



כיצד נבחן את equals()?
פשוט נכתוב גם עבורה תוכנית בדיקה, שנוסיף למחלקת הבדיקה.

```
public void testEquals() {
    Money m12CHF= new Money(12, "CHF");
    Money m14CHF= new Money(14, "CHF");

    Assert.assertTrue(!m12CHF.equals(null));
    Assert.assertEquals(m12CHF, m12CHF);
    Assert.assertEquals(m12CHF, new Money(12, "CHF")); // (1)
    Assert.assertTrue(!m12CHF.equals(m14CHF));
}
```

לאחר הרצה, נקבל את המבט הבא: כאשר ניתן לראות את שני הטסטים שבוצעו, ובהצלחה.



לעיתים קרובות אנו בונים את אוסף האוביקטים המשמשים אותנו לטסט, ועושים בו שימוש לאורך כמה test cases. במקרה זה, הדבר יהיה יעיל יותר, אם נגדיר מראש את אוסף האוביקטים לפני התחלת הרצת הטסטים.

אם שמתם לב, מתודת ה setup() בחלון יצירת ה TestCase סומנה ב - √ , לכן תבנית המתודה כבר הוכנה לנו.

עדכנו את מחלקת הטסט כך שתשקף את הקוד הבא:

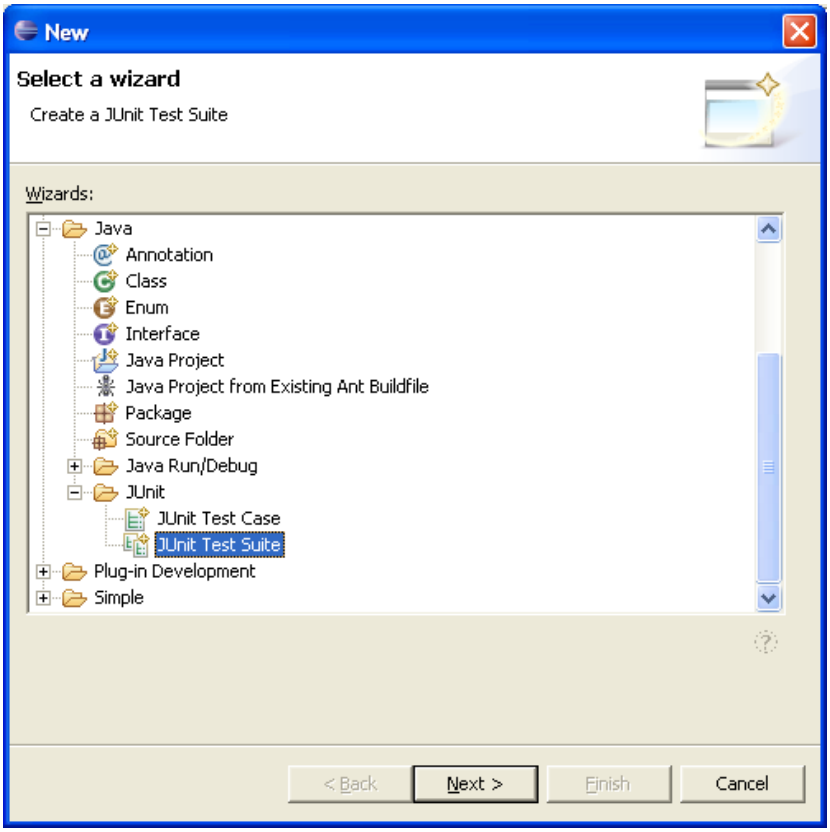
```
public class MoneyTest extends TestCase {
…
    private Money f12CHF;
    private Money f14CHF;

    protected void setUp() {
        f12CHF= new Money(12, "CHF");
        f14CHF= new Money(14, "CHF");
    }
…
}
```

הוציאו את ההצהרות על משתנים אלה מתוך מתודות הטסט.
והריצו שוב, לוודא שהתוצאה תקינה.

כעת נסתכל על מחלקה נוספת שיכולה להכיל הרבה טסטים. זוהי מחלקת ה TestSuite שממנה ניתן לנהל הרצה של הרבה TestCases.

יצרו בפרויקט מחלקה חדשה שיורשת מ Test Suite. עשו זאת דרך file->new->other->Junit

ובחרו ב JUnit Test Suite.



עם בקשת יצירת המחלקה, יפתח אשף שישאל אתכם, אילו טסטים אתם רוצים לכלול בחבילה. סמנו את הטסטים הרצויים (כרגע יש לכם TestCase אחד) ואשרו.

כדי להריץ את החבילה מתוך Eclipse Run->run as-> Junit

הקוד שנוצר הינו קוד דינמי שטוען את כל הטסטים מהמחלקה הנתונה.

```
public class AllTests {

	public static Test suite() {
		TestSuite suite = new TestSuite("Test for default package");
		suite.addTestSuite(MoneyTest.class);
		return suite;
	}

}
```

ניתן בצורה סטטית לטעון את כל הטסטים שכתבנו בצורה הבאה:

```
public static Test suite() {
    TestSuite suite= new TestSuite();
    suite.addTest(
        new MoneyTest("money equals") {
            protected void runTest() { testEquals(); }
        }
```

```
    );

    suite.addTest(
        new MoneyTest("simple add") {
            protected void runTest() { testSimpleAdd(); }
        }
    );
    return suite;
}
```

בשלב הבא נעבור להשתמש בחשבונות כספים המכילים מטבעות שונים.

הוסף את המחלקה MoneyBag לפרויקט

```
class MoneyBag {
    private Vector fMonies= new Vector();

    MoneyBag(Money m1, Money m2) {
        appendMoney(m1);
        appendMoney(m2);
    }

    MoneyBag(Money bag[]) {
        for (int i= 0; i < bag.length; i++)
            appendMoney(bag[i]);
    }
}
```

ממש את המתודות `appendMoney` ו equals

צור מחלקת בדיקות חדשה והוסף לה את ה fixtures הבאים תחת מתודת ה setup

```
protected void setUp() {
    f12CHF= new Money(12, "CHF");
    f14CHF= new Money(14, "CHF");
    f7USD=  new Money( 7, "USD");
    f21USD= new Money(21, "USD");
    fMB1= new MoneyBag(f12CHF, f7USD);
    fMB2= new MoneyBag(f14CHF, f21USD);
}
```

כעת השתמשו בהם לבדיקה הבאה:

```
public void testBagEquals() {
    Assert.assertTrue(!fMB1.equals(null));
    Assert.assertEquals(fMB1, fMB1);
    Assert.assertTrue(!fMB1.equals(f12CHF));
    Assert.assertTrue(!f12CHF.equals(fMB1));
    Assert.assertTrue(!fMB1.equals(fMB2));
}
```

כעת אנו יכולים לעדכן את add(Money A) כך שתוכל להוסיף סכום כסף ממטבע אחר ולהחזיר חשבון כספים.

```
public Money add(Money m) {
    if (m.currency().equals(currency()) )
        return new Money(amount()+m.amount(), currency());
    return new MoneyBag(this, m);
}
```

העדכון האחרון לא יעבור הידור, כי המתודה מצפה להחזיר אוביקט מסוג Moneyולא MoneyBag.
ע"י הגדרת מנשק IMoney ששתי המחלקות יורשות ממנו ניתן יהיה להמשיך.
עדכון המימוש בשתי המחלקות מופיע בתיקיה FullMoneyExample.

הוסף את בדיקות ההשוואה לחשבונות שונים

```
public void testBagEquals() {
    Assert.assertTrue(!fMB1.equals(null));
    Assert.assertEquals(fMB1, fMB1);
    Assert.assertTrue(!fMB1.equals(f12CHF));
    Assert.assertTrue(!f12CHF.equals(fMB1));
    Assert.assertTrue(!fMB1.equals(fMB2));
}
```

```
public void testMixedSimpleAdd() {
    // [12 CHF] + [7 USD] == {[12 CHF][7 USD]}
    Money bag[]= { f12CHF, f7USD };
    MoneyBag expected= new MoneyBag(bag);
    Assert.assertEquals(expected, f12CHF.add(f7USD));
}
```

נקודה נוספת שצריך לשים לב אליה: אם נתון לנו חשבון (MoneyBag) המכיל שני מטבעות שונים ונחסר מהחשבון בדיוק את הסכום של שני המטבעות, הרי שהתוצאה שווה לאוביקט Money.

הוסיפו את הבדיקה הבאה שמוודאת זאת לרשימת הטסטים

```
public void testSimplify() {
    // {[12 CHF][7 USD]} + [-12 CHF] == [7 USD]
    Money expected= new Money(7, "USD");
    Assert.assertEquals(expected, fMB1.add(new Money(-12, "CHF")));
}
```

עבודה נעימה ☺

מקורות: eclipse.org ,junit.sourceforge.net