JUnit

JUnit

- Unit testing för java
- Används för att testa att metoder/klasser beter sig som det var tänkt
- Många IDE:er tex Eclipse har inbyggt stöd för detta.

JUnit 3

- Vi skriver testklasser och låter dessa ärva från `junit.framework.TestCase`. (JUnit klasserna måste läggas till till projektet då dessa klasser inte är med bland standardklasserna i java)
- Namnge testmetoderna med test som prefix (det är så de känns igen som testmetoder)
- Undersök de villkor som ska testas mha de olika assert-metoderna i `TestCase`
- Kör testen mha Run -> Run… -> JUnit i Eclipse (Se till så att ni ställt in den på att använda version 3)

JUnit 3 forts.

- Ibland vill man göra grundinställningar som ska göras innan varje test. Dessa kan göras i metoden `protected void setUp()` som körs innan varje test i den klassen
- Behöver man städa upp efter testen görs detta i metoden `protected void tearDown()` som anropas automatiskt efter varje test.

Villkorskontroller i test

- Statiska metoder som finns definierade i Klassen Assert (TestCase ärver från denna)
  - `assertTrue`
  - `assertFalse`
  - `assertNull`
  - `assertNotNull`
  - `assertEquals`
    - Kontrollerar om två **värden** är lika. För objekt kontrolleras detta mha deras equals-metod.
  - `assertSame`
    - Kontrollerar om två **referenser** är lika
  - `assertNotSame`
  - `fail()`
    - Misslyckas alltid

Exempel JUnit3

```
import junit.framework.TestCase;

public class PolynomTest extends TestCase {
   public void testCreateObject() {
      double d[]={1.0,2.0};
      Polynom p=new Polynom(d);
      assertNotNull(p);
   }
}
```

JUnit 4

- Version 4
  - Vi behöver ej utnyttja arv
  - Testmetoder indikeras mha att vi annoterar dem med @Test
  - Till Test-annotationen kan man även specificera om vi vill att ngt särskilt undantag ska kastas
    `@Test(expected=IndexOutOfBoundsException.class) public void outOfBounds() { new ArrayList<Object>().get(1); }`
    eller sätta en tidsgräns för testet
    `@Test(timeout=100)`
    Här 100 millisekunder
  - Använd assert-metoderna för att testa villkoren (samma som i JUnit 3) Ett lätt sätt att använda assert-metoderna (eftersom de nu inte är tillgängliga via arv) är att göra en `static import` från Assert-klassen

Static imports

```
public class Demo7
{
    public static void main( String[] argv )
    {
        System.out.println( 2 * Math.PI );
    }
}
```

Bra? Nja ....
Kräver viss försiktighet

```
import static java.lang.Math.PI;

public class Demo7
{
    public static void main( String[] argv )
    {
        System.out.println( 2 * PI );
    }
}
```

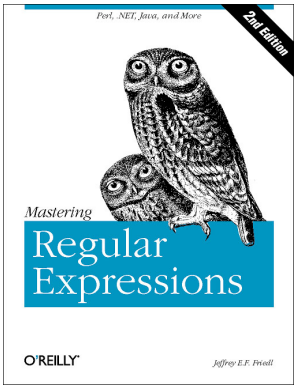
Exempel med JUnit4

```
import org.junit.Test;
import static junit.framework.Assert.assertNotNull;

public class JUnit4TestClass {
   @Test public void createObject() {
      double d[]={1.0,2.0};
      Polynom p=new Polynom(d);
      assertNotNull(p);
   }
}
```

Reguljära uttryck

Reguljära uttryck

- Ett sätt att söka och manipulera text
- Konstruerar mönster som försöker hitta nåt i texten
- Viktigt, användbart, många resurser
  - http://regex.info/
  - http://etext.lib.virginia.edu/services/helpsheets/unix/regex.html
  - http://www.regular-expressions.info/
  - http://regexlib.com/default.aspx
  - http://regexlib.com/cheatsheet.aspx



```
Boka en resa för boken som finns
i bokstugan norr om bokrike
```

```
bok(en|stugan|rike)
```

```
bock\1
```

```
Boka en resa för bocken som finns
i bockstugan norr om bockrike
```



```
Nu 17 kommer en siffra78 och en
till 78 men utan 7kstuga 89
```

```
\b(\d+)\b
```

```
SIFFRA(\1)
```

```
Nu SIFFRA(17) kommer en siffra78
och en till SIFFRA(78) men utan
7kstuga SIFFRA(89)
```



```
import java.util.regex.*;

public class Demo01
{
    private static final String PatternToLookFor = "\\b(\\d+)\\b";
    private static final String TargetString =
        "Nu kommer en siffra78 och en till 78 men utan 7kstuga 89";

    public static void main( String args[] )
    {
        Pattern p = Pattern.compile( PatternToLookFor );
        Matcher m = p.matcher( TargetString );
        System.out.println( m.replaceAll( "SIFFRA($1)" ) );
    }
}
```

Nu kommer en siffra78 och en till SIFFRA(78)
men utan 7kstuga SIFFRA(89)


```
import java.util.regex.*;

public class Demo02
{
    private static final String PatternToLookFor = "\\b(\\d+)\\b";
    private static final String TargetString =
        "Nu kommer en siffra78 och en till 78 men utan 7kstuga 89";

    public static void main( String args[] )
    {
        Pattern p = Pattern.compile( PatternToLookFor );
        Matcher m = p.matcher( TargetString );

        while( m.find() ){
            System.out.println( m.group( ) + "   start " +
                m.start( ) + "   end " + m.end( ) );
        }
    }
}
```

```
> java Demo02
78   start 34   end 36
89   start 54   end 56
```



```
This is a short demo string
    that can be used by those who wants to play
        with some regular    expressions. Regular
expressions,     or as they sometimes are called,
"regexps" are useful. What
can you say: wiff,wiff perhaps or between
words
```



```
import java.io.*;
import java.util.regex.*;

public class Demo03
{
   private final static String EXAMPLE = "example.txt";
   private final static String PATTERN = "reg";

   private static String getText( )
   {
      char   inp[]  = new char[1000];
      try{
         Reader input  = new BufferedReader( new FileReader( EXAMPLE ) );
         input.read( inp );
      } catch ( java.io.FileNotFoundException fnf ) {
         fnf.printStackTrace();
      } catch ( java.io.IOException io ){
         io.printStackTrace();
      }
      return new String(inp);
   }

   public static void main( String[] argv )
   {
        Pattern p = Pattern.compile( PATTERN );
        Matcher m = p.matcher( getText() );

        while( m.find() ){
            System.out.println( m.group( ) );
        }
   }
}
```

Hitta alla ord dom börjar på ‘w’

```
"w.*"
```

```
> java Demo03
who wants to play
with some regular    expressions. Regular
wiff,wiff perhaps or between
words
```


Hitta alla ord dom börjar på ‘w’

```
"w[^ ]*"
```

```
> java Demo03
who
wants
with
wiff,wiff
ween
words
```


Hitta alla ord dom börjar på ‘w’

```
" w[^ ]*"
```

```
> java Demo03
who
wants
with
wiff,wiff
```


Hitta alla ord dom börjar på ‘w’

```
" w\\w*"
```

```
> java Demo03
 who
 wants
 with
 wiff
```


Hitta alla ord dom börjar på ‘w’

```
"\\bw\\w*\\b"
```

```
> java Demo03
who
wants
with
wiff
wiff
words
```


\A	Början av en sträng
\b	Ordgräns
\B	Inte en ordgräns
\d	Siffra
\D	Inte en siffra
\s	whitespace
\S	Inte whitespace
\t	Tab
\w	“ordtecken”
\W	Inte “ordtecken”
\z	Slut på sträng
\Z	Slut på sträng eller före nyrad

*	0 eller flera gånger
+	1 eller flera gånger
?	0 eller 1
{count}	Exact ‘count’ gånger
{min,}	Minst ‘min’ gånger
{min,max}	min ≤ x ≤ max
*?	0 eller fler (minimalt)
+?	1 eller fler (minimalt)
??	0 eller 1 (minimalt)
{min,}?	Minst ‘min’ gånger (minimalt)
{min,max}?	min ≤ x ≤ max (minimalt)

Ta bort mellanslag i början på raden

```
public static void main( String[] argv )
{
     Pattern p = Pattern.compile( PATTERN );
     Matcher m = p.matcher( getText() );

     System.out.println( m.replaceAll( "" ) );
}
```

"^\\s*"

```
> java Demo04
This is a short demo string
    that can be used by those who wants to play
          with some regular    expressions. Regular
expressions,     or as they sometimes are called,
"regexps" are useful. What
can you say: wiff,wiff perhaps or between
words
```


Ta bort mellanslag i början på raden

```
public static void main( String[] argv )
{
     Pattern p = Pattern.compile( PATTERN, Pattern.MULTILINE );
     Matcher m = p.matcher( getText() );

     System.out.println( m.replaceAll( "" ) );
}
```

"^\\s*"

```
 > java Demo05
This is a short demo string
that can be used by those who wants to play
with some regular    expressions. Regular
expressions,     or as they sometimes are called,
"regexps" are useful. What
can you say: wiff,wiff perhaps or between
words
```


Ta bort ny rad och bara ett mellanslag

"\\n|\\s+"

```
> java Demo06
This is a short demo string  that can be used
by those who wants to play  with some regular
expressions. Regular expressions, or as they
sometimes are called, "regexps" are useful.
What can you say: wiff,wiff perhaps or between
words
```


Ta bort ny rad och bara ett mellanslag

"\\n\\s*|\\s+"

```
 > java Demo06
This is a short demo string that can be used by
those who wants to play with some regular
expressions. Regular expressions, or as they
sometimes are called, "regexps" are useful.
What can you say: wiff,wiff perhaps or between
words
```


Enklare variant ...

```
public class Demo07
{
    private static final String PatternToLookFor = "\\b(\\d+)\\b";
    private static final String TargetString =
        "Nu kommer en siffra78 och en till 78 men utan 7kstuga 89";

   public static void main( String[] argv )
   {
        System.out.println( TargetString.replaceAll( PatternToLookFor,
          "SIFFRA($1)" ) );
   }
}
```

```
> java Demo07
Nu kommer en siffra78 och en till
SIFFRA(78) men utan 7kstuga SIFFRA(89)
```

Diverse

Variabelt antal argument

```
public class Demo5
{
    public static void testing( String name, Integer ... xf )
    {
        System.out.println( name );
        for( Integer x : xf ){
            System.out.println( x );
        }
        System.out.println( name + " slut" );
    }

    public static void main(String[] args)
    {
        testing( "Some name", 1, 2, 3, 4 );
        testing( "Ettan", 1, 2 );
        testing( "miffo", "mju", 1, 2 );
    }
}
```

```
> java Demo5
Some name
1
2
3
4
Some name slut
Ettan
1
2
Ettan slut
```

Formatted output

```
public class Demo6
{
    public static void main(String[] args)
    {
        System.out.printf("%3d > %2.3f %s\n", 12, 3.14159, "playing" );
        System.out.printf("%9d > %2.1f %s\n", 12, 3.14159, "playing" );
        System.out.printf("%-9d > %2.6f %s\n", 12, 3.14159, "playing" );

    }
}
```

```
> java Demo6
 12 > 3.142 playing
       12 > 3.1 playing
12        > 3.141590 playing
>
```

Paket

- Ett Java paket är en mängd klasser
- Paket användas för att gruppera liknande klasser och/eller sådana som beror på varandra
- Klasserna i ett paket behöver inte ärva från varandra
- Java `util` är ett paket (`java.util`)
- Java API:n består av många paket
- `import` satsen gör innehållet i ett paket tillgängligt

```
import paketnamn.klassnamn;   ------- enstaka klass
import paketnamn.*;           ....................... alla klasser i paketet
```


Enumeration Repetition

Enumeration

- Skandal att detta inte fanns i Java 1.0
- Fungerar som i andra språk

```
public class Navigate
{
    final public static int NORTH = 1;
    final public static int SOUTH = 2;
    final public static int EAST  = 3;
    final public static int WEST  = 4;

    void go( int dir )
    {
        switch( dir ){
            case NORTH:
                // ...
                break;
            case SOUTH:
                // ...
                break;
            case WEST:
                // ...
                break;
            case EAST:
                // ...
                break;
            default:
                // ...
        }
    }
}
```

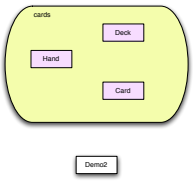


```
public class Navigate
{
    public enum Direction { NORTH, SOUTH, EAST, WEST };

    void go( Direction dir )
    {
        switch( dir ){
            case NORTH:
                // ...
                break;
            case SOUTH:
                // ...
                break;
            case WEST:
                // ...
                break;
            case EAST:
                // ...
                break;
        }
    }
}
```



```
public class Card
{
    public enum Rank { DEUCE, THREE, FOUR, FIVE, SIX, SEVEN, EIGHT,
        NINE, TEN, JACK, QUEEN, KING, ACE }
}


public class Demo1
{
    public static void main( String[] argv )
    {
        Card.Rank r = Card.Rank.FIVE;
        System.out.println( r );
    }
}
```

```
> java Demo1
FIVE
```


```
import cards.Deck;
import cards.Card;

public class Demo2
{
    public static void main( String[] args )
    {
        int numHands = Integer.parseInt( args[0] );
        int handSize = Integer.parseInt( args[1] );

        Deck d = Deck.newDeck();

        for( int i = 0; i < numHands; i++ ){
            System.out.println( d.getHand( handSize) );
        }
    }
}
```

```
> java Demo2 4 5
[JACK of CLUBS, THREE of CLUBS, FOUR of SPADES, TEN of HEARTS, FIVE of DIAMONS]
[QUEEN of CLUBS, KING of CLUBS, THREE of DIAMONS, EIGHT of CLUBS, FIVE of SPADES]
[THREE of SPADES, NINE of CLUBS, SEVEN of HEARTS, FIVE of HEARTS, EIGHT of DIAMONS]
[SEVEN of CLUBS, SIX of SPADES, NINE of DIAMONS, JACK of SPADES, DEUCE of HEARTS]
```



```
package cards;

public class Card
{
    public enum Rank { DEUCE, THREE, FOUR, FIVE, SIX, SEVEN, EIGHT,
        NINE, TEN, JACK, QUEEN, KING, ACE }

    public enum Suit { CLUBS, DIAMONS, HEARTS, SPADES }

    private final Rank rankValue;
    private final Suit suitValue;

    Card( Rank rankValue, Suit suitValue )
    {
        this.rankValue = rankValue;
        this.suitValue = suitValue;
    }

    public Rank rank( )
    {
        return rankValue;
    }

    public Suit suit( )
    {
        return suitValue;
    }

    public String toString( )
    {
        return rankValue + " of " + suitValue;
    }
}
```

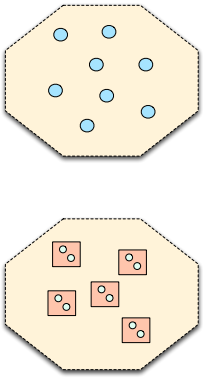
```
package cards;

import java.util.List;
import java.util.ArrayList;

public class Hand
{
    private List<Card> hand;

    Hand( List<Card> content )
    {
        hand = content;
    }

    public String toString( )
    {
        return hand.toString();
    }
}
```



```
package cards;

import java.util.ArrayList;
import java.util.Collections;
import java.util.List;

public class Deck
{
    private static final List<Card> protoDeck = new ArrayList<Card>();

    private List<Card> myDeck;

    private Deck( )
    {
        myDeck = new ArrayList<Card>( protoDeck );
        Collections.shuffle( myDeck );
    }

    public Hand getHand( int nrOfCards )
    {
        List<Card> handCards = myDeck.subList( 0, nrOfCards );
        Hand hand            = new Hand( new ArrayList<Card>(handCards) );
        handCards.clear();
        return hand;
    }

    public static Deck newDeck( )
    {
        return new Deck();
    }

    static{
        for( Card.Suit suit : Card.Suit.values( ) ){
            for( Card.Rank rank : Card.Rank.values( ) ){
                protoDeck.add( new Card( rank, suit ) );
            }
        }
    }
}
```

Inte en klass utan en enum!!!

```
public enum Planet
{
    MERCURY (3.303e+23, 2.4397e6),
    VENUS   (4.869e+24, 6.0518e6),
    EARTH   (5.976e+24, 6.37814e6),
    MARS    (6.421e+23, 3.3972e6),
    JUPITER (1.9e+27,   7.1492e7),
    SATURN  (5.688e+26, 6.0268e7),
    URANUS  (8.686e+25, 2.5559e7),
    NEPTUNE (1.024e+26, 2.4746e7),
    PLUTO   (1.27e+22,  1.137e6);

    private final double mass;   // in kilograms
    private final double radius; // in meters

    Planet(double mass, double radius)
    {
        this.mass = mass;
        this.radius = radius;
    }

    public double mass()   { return mass; }
    public double radius() { return radius; }

    // universal gravitational constant  (m3 kg-1 s-2)
    public static final double G = 6.67300E-11;

    public double surfaceGravity()
    {
        return G * mass / (radius * radius);
    }
    public double surfaceWeight(double otherMass)
    {
        return otherMass * surfaceGravity();
    }
}
```




```
public class Demo3
{
    public static void main(String[] args)
    {
        double earthWeight = Double.parseDouble(args[0]);
        double mass        = earthWeight/Planet.EARTH.surfaceGravity();

        for( Planet p : Planet.values() )
            System.out.printf("Your weight on %s is %f%n",
                              p, p.surfaceWeight(mass));
    }
}
```



```
public class Demo4
{
    public enum Operation{
        PLUS   { double eval( double x, double y) { return x + y; } },
        MINUS  { double eval( double x, double y) { return x - y; } },
        TIMES  { double eval( double x, double y) { return x * y; } },
        DIVIDE { double eval( double x, double y) { return x / y; } };

        abstract double eval( double x, double y);
    }

    public static void main(String[] args)
    {
        double x = Double.parseDouble(args[0]);
        double y = Double.parseDouble(args[1]);

        for( Operation op : Operation.values() ){
            System.out.println( x + " " + op + " " + y + " = " +
                op.eval(x,y));
        }
    }
}
```

```
> java Demo4 1 2
1.0 PLUS 2.0 = 3.0
1.0 MINUS 2.0 = -1.0
1.0 TIMES 2.0 = 2.0
1.0 DIVIDE 2.0 = 0.5
```