Nye features i Java 1.5/5.0

Denne artikel beskriver nye features i Java 1.5/5.0, som lige er kommet i final (ikke-beta) udgave.

Den forudsætter et vist kendskab til Java.

Skrevet den **14. Feb 2010** af **arne_v** I kategorien **Programmering / Java** | ★★★☆☆

Historie:
V1.0 - 11/02/2004 - original
V1.1 - 18/02/2004 - præcisere at generics kan bruges til andet end type safe collections
V1.2 - 29/10/2004 - en masse rettelser efter release (versions nummer etc.)
V1.3 - 05/11/2004 - tilføj beskrivelse af Scanner og System.getenv
V1.4 - 05/08/2005 - tilføj StringBuilder, thread pool m.m.
V1.5 - 26/12/2008 - små ændringer og tilføj links
V1.6 - 14/02/2010 - smårettelser

Kodenavn Tiger

Java 1.5/5.0 med kodenavnet Tiger udkom i september 2004.

Lad os starte med version nummer forvirringen. SUN har af uransagelige årsager valgt ikke at release den som version 1.5 men som version 5.0.

Tåbelig beslutning efter min mening. Men de vil tilsyneladende konkurrere med Microsoft om at have højeste versions nummer. Jeg mener at de fleste udviklere ligger mere vægt på produktet end på versions nummeret.

Som min stille protest vil jeg kalde det 1.5 i resten af artiklen.

Denne version har været længe ventet, fordi den vil udvide ikke bare Java API men også selve Java Language med en masse nye features.

Og SUN har snakket om disse i næsten et år.

Den kan downloades fra http://java.sun.com/javase/downloads/index.jsp.

Husk når man compiler at bruge:
-source 1.5
og evt.:
-target 1.5
for at få alle de nye muligheder med.

Hvis man skal compile til Microsoft ældgamle JVM (som stadig er JVM i mnage Internet Explorer) så skal man bruge:
-source 1.3 -target 1.1

Static import

I Java 1.5 kan man lave en "import static" af en klasse så man kan undgå at prefixe static metoder og variable med klassens navn.

Eksempler:

SI14.java

```
import java.lang.Math;

public class SI14 {
   public static void main(String[] args) {
       System.out.println(Math.pow(2, 6));
   }
}
```

(jeg ved godt at den import ikke er nødvendig)

SI15.java

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

public class SI15 {
   public static void main(String[] args) {
       System.out.println(pow(2, 6));
   }
}
```

SIX14.java

```
import java.awt.BorderLayout;
import javax.swing.JFrame;
import javax.swing.JLabel;

public class SIX14 extends JFrame {
    public SIX14() {
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        getContentPane().setLayout(new BorderLayout());
        getContentPane().add(new JLabel("N"), BorderLayout.NORTH);
        getContentPane().add(new JLabel("W"), BorderLayout.WEST);
        getContentPane().add(new JLabel("E"), BorderLayout.EAST);
        getContentPane().add(new JLabel("S"), BorderLayout.SOUTH);
        pack();
    }
    public static void main(String[] args) {
```

```
        SIX14 f = new SIX14();
        f.setVisible(true);
    }
}
```

SIX15.java

```
import java.awt.BorderLayout;
import javax.swing.JFrame;
import javax.swing.JLabel;

import static java.awt.BorderLayout.*;

public class SIX15 extends JFrame {
    public SIX15() {
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        getContentPane().setLayout(new BorderLayout());
        getContentPane().add(new JLabel("N"), NORTH);
        getContentPane().add(new JLabel("W"), WEST);
        getContentPane().add(new JLabel("E"), EAST);
        getContentPane().add(new JLabel("S"), SOUTH);
        pack();
    }
    public static void main(String[] args) {
        SIX15 f = new SIX15();
        f.setVisible(true);
    }
}
```

Nyttig feature men ikke specielt vigtig efter min mening.

Auto boxing/unboxing

Java 1.5 kan konvertere mellem simple data typer og deres tilsvarende objekt automatisk.

Denne feature hedder auto boxing og auto unboxing. Og den er lånt fra C#.

Eksempel:

ABUB14.java

```
public class ABUB14 {
    public static void main(String[] args) {
        int i1 = 123;
        Integer i2 = new Integer(i1);
        System.out.println(i2);
```

```
        int i3 = i2.intValue();
        System.out.println(i3);
    }
}
```

ABUB15.java

```
public class ABUB15 {
    public static void main(String[] args) {
        int i1 = 123;
        Integer i2 = i1;
        System.out.println(i2);
        int i3 = i2;
        System.out.println(i3);
    }
}
```

En særdeles nyttig feature som kan hjælpe til at gøre koden mere læselig.

Generics

Java 1.5 har mulighed for at lave klasser/interfaces som indeholder vilkårlige typer.

Denne feature kaldes generics. Og den er lånt fra C++ (templates).

Den kan bl.a. lave type safe collections.

Eksempel:

G14.java

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

public class G14 {
    public static void main(String[] args) {
        List lst = new ArrayList();
        lst.add(new Integer(1));
        lst.add(new Integer(22));
        lst.add(new Integer(333));
        // denne vil give en runtime fejl:
        // lst.add("abc");
        for(int i = 0; i < lst.size(); i++) {
            Integer v = (Integer)lst.get(i);
            System.out.println(v);
        }
```

```
    }
}
```

G15.java

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

public class G15 {
    public static void main(String[] args) {
        List<Integer> lst = new ArrayList<Integer>();
        lst.add(new Integer(1));
        lst.add(new Integer(22));
        lst.add(new Integer(333));
        // denne vil give en compile fejl:
        // lst.add("abc");
        for(int i = 0; i < lst.size(); i++) {
            Integer v = lst.get(i);
            System.out.println(v);
        }
    }
}
```

En yderst nyttig feature som kan gøre koden meget mere runtime sikker.

Bemærk at generics også kan anvendes udover collections, men det er bare der hvor man forventer det mest brugt.

Ny for løkke

I Java 1.5 kan man erstatte:

```
Iterator it = xxx.iterator();
while(it.hasNext()) {
   X v = (X)it.next();
   ...
}
```

med:

```
for(X v : xxx) {
   ...
}
```

Denne feature kendes fra flere andre sprog som foreach løkke.

Eksempel som også bruger generics og auto boxing/unboxing:

F14.java

```
import java.util.HashMap;
import java.util.Iterator;
import java.util.Map;

public class F14 {
    public static void main(String[] args) {
        Map m = new HashMap();
        m.put(new Integer(1), "a");
        m.put(new Integer(2), "bb");
        m.put(new Integer(3), "ccc");
        Iterator it = m.keySet().iterator();
        while(it.hasNext()) {
            Integer k = (Integer)it.next();
            String v = (String)m.get(k);
            System.out.println(k + " " + v);
        }
    }
}
```

F15.java

```
import java.util.HashMap;
import java.util.Iterator;
import java.util.Map;

public class F15 {
    public static void main(String[] args) {
        Map<Integer,String> m = new HashMap<Integer,String>();
        m.put(1, "a");
        m.put(2, "bb");
        m.put(3, "ccc");
        for(int k : m.keySet()) {
            String v = m.get(k);
            System.out.println(k + " " + v);
        }
    }
}
```

Mindre nyttig feature i sig selv men meget nyttig sammen med generics og auto boxing/unboxing.

Type safe enums

Java 1.5 understøtter nu enum men modsat C, C++ og C# i en type safe form.

Eksempel:

E14std.java

```
public class E14std {
    public static final int C = 1;
    public static final int CPP = 2;
    public static final int CS = 3;
    public static final int JAVA = 4;
    public static String toString(int lang) {
        switch(lang) {
            case C:
                return "C";
            case CPP:
                return "CPP";
            case CS:
                return "CS";
            case JAVA:
                return "JAVA";
            default:
                return "Unknown";
        }
    }
    public static void main(String[] args) {
        int lang = E14std.JAVA;
        System.out.println(E14std.toString(E14std.JAVA));
    }
}
```

E14smart.java

```
public class E14smart {
    private String name;
    private E14smart(String name) {
        this.name = name;
    }
    public String toString() {
        return name;
    }
    public static final E14smart C = new E14smart("C");
    public static final E14smart CPP = new E14smart("CPP");
    public static final E14smart CS = new E14smart("CS");
    public static final E14smart JAVA = new E14smart("JAVA");
    public static void main(String[] args) {
        E14smart lang = E14smart.JAVA;
        System.out.println(lang);
    }
```

```
}
```

Language.java

```
public enum Language { C, CPP, CS, JAVA };
```

E15.java

```
public class E15 {
    public static void main(String[] args) {
        Language lang = Language.JAVA;
        System.out.println(lang);
    }
}
```

En feature som kan være meget nyttig i nogle tilfælde.

Scanner

Java 1.5 har fået en utility klasse java.util.Scanner til at parse input med.

Eksempel:

S14.java

```
import java.io.*;

public class S14 {
    public static void main(String[] args) throws Exception {
        BufferedReader br = new BufferedReader(new
InputStreamReader(System.in));
        System.out.print("Indtast heltal: ");
        int iv = Integer.parseInt(br.readLine());
        System.out.println(iv);
        System.out.print("Indtast decimaltal: ");
        double xv = Double.parseDouble(br.readLine());
        System.out.println(xv);
        System.out.print("Indtast ord: ");
        String sv = br.readLine();
        System.out.println(sv);
    }
}
```

S15.java

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

public class S15 {
    public static void main(String[] args) throws Exception {
        Scanner scn = new Scanner(System.in);
        System.out.print("Indtast heltal: ");
        int iv = scn.nextInt();
        System.out.println(iv);
        System.out.print("Indtast decimaltal: ");
        double xv = scn.nextDouble();
        System.out.println(xv);
        System.out.print("Indtast ord: ");
        String sv = scn.next();
        System.out.println(sv);
    }
}
```

En feature som kan være meget nyttig i nogle tilfælde.

Varargs

Java 1.5 understøtter variabelt antal argumenter som kendt fra C, C++ og C#.

Eksempel:

VA.java

```
public class VA {
    public static void doit(Object ... v) {
        for(int i = 0; i < v.length; i++) {
            System.out.print(" " + v[i]);
        }
        System.out.println();
    }
    public static void main(String[] args) {
        doit(1);
        doit(1, 2);
        doit(1, 2, 3);
    }
}
```

Meget praktisk feature efter min mening.

Formatted output

Java 1.5 supporterer nu formateret output via printf som vi kender den fra C.

Eksempel:

FIO.java

```
public class FIO {
    public static void main(String[] args) {
        int iv = 123;
        double xv = 123.456;
        String sv = "abc";
        System.out.printf("%d %f %s\n", iv, xv, sv);
    }
}
```

Ikke så vigtig en feature efter min mening.

System.getenv

Java 1.5 har reaktiveret System.getenv, som blev deaktiveret i 1.2.

Eksempel:

GE.java

```
public class GE {
    public static void main(String[] args) throws Exception {
        System.out.println(System.getenv("JAVA_HOME"));
    }
}
```

StringBuilder

Java 1.5 har suppleret/erstattet StringBuffer klassen med en ny StringBuilder klasse.

Deres relationship er den samme som mellem Hashtable og HashMap.

Den nye er ikke thread safe og derfor en lille smule hurtigere

Jeg synes ikke at det er en speciel vigtig ændring. Og jeg vil godt spå at det vil tage 10 år at slippe af med StringBuffer.

Samtidigheds utilities

Java 1.5 har fået en pakke java.util.concurrent med mange spændende ting i.

Jeg vil ikke komme med eksempler på funktionaliteten, da det vil tage for meget plads, men jeg vil kort nævne nogle af de klasser som jeg finder spændende.

BlockingQueue interface:

add metode som blocker indtil der er plads i køen
take metode som blocker indtil der er noget i køen
ArrayBlockingQueue class med en fast max. kø længde
LinkedBlockingQueue class med uendelig max. kø længde

Semaphore class:

constructor som angiver antal samtidige
acquire metode som blocker indtil der er plads
release metode

Der er også:
  ExecutorService, Caller og Future interfacene
  ScheduledThreadPoolExecutor og ThreadPoolExecutor klasserne
som implementerer en komplet thread pool.

Jeg tror at der er rigtigt mange spændende ting i den pakke, som man kan få stor nytte af.

Annotations

Java 1.5 har support for metadata i koden kalde annotations ligesom C# attributter.

Den feature henvender sig primært til værktøjer som genererer dynamiske proxies, AOP frameworks og den slags.

Men der er en enkelt nyttig mulighed for almindelige programmører.

Hvis du erklærer en metode som:

```
@Override
public void dosomething() {
  ...
}
```

så får du fejl, hvis dosomething ikke overrider en metode i super klassen.

Det er smart til at undgå stave fejl, hvor man tror at man har overridet en metode, men det har man slet ikke.

Jeg vil ikke give yderligere eksempler på brug af dette.

Class data sharing

Java 1.5 mapper readonly data i classes til shared memory, hvilket kan spare hukommelse, når man kører flere JVM's på samme maskine.

JMX

Java 1.5 includerer et framework for management kaldet JMX.

Jeg vil ikke give eksempler på brug af dette.

Java 1.6/6.0

Se artikel:
http://www.eksperten.dk/guide/941 "Nye features i Java 1.6/6.0"

Kommentar af mora d. 04. May 2004 | 1

Kommentar af simonvalter d. 13. Feb 2004 | 2

Endnu en god og interessant artikel

Kommentar af bearhugx d. 13. Feb 2004 | 3

En interesant artikel, som har især givet mig lyst til at gå i krig med afprøvning af Java 1.5 :-) - Godt skrevet - og velafgrænset. - Gode eksempler - hvor man både ser "old-school"-måden og den tilsvarende nye måde at kode på :-) -- Ikke andet at sige en dobbelplusgod!

Kommentar af jnh d. 07. Nov 2005 | 4

Den var alle 5 point værd... Burde koste 100 ca.

Kommentar af daniboy d. 06. Apr 2005 | 5

Fed artikkel :)
Rettelse?
Java 1.5 har fået en pakke java,util.concurrent med mange spændende ting i.
Burde værre
Java 1.5 har fået en pakke java.util.concurrent med mange spændende ting i.

Kommentar af conrad d. 12. Feb 2004 | 6

Kommentar af soreno d. 04. Apr 2004 | 7

Glimrende opslagsværk til hurtigt at kunne sætte sig ind i syntaksen af nye features som JDK v1.5 bringer.

Kommentar af michaeltajo d. 08. Nov 2005 | 8

Kort og godt

Kommentar af viciodk d. 11. Feb 2004 | 9

Utrolig god artikel! :)

Kommentar af mikkelbm d. 30. Nov 2004 | 10

Super

Kommentar af baxos d. 12. Feb 2004 | 11

Kommentar af william_munny d. 14. Feb 2004 | 12

fin artikel med gode eksempler.

Kommentar af oo_soeren d. 12. Nov 2004 | 13

Glimrende artikel, der på en lettilgængelig måde pinpointer nye features i Java - med gode eksempler.

Kommentar af schwarz84 d. 11. Aug 2005 | 14

God artikel. I øvrigt kan foreach-løkken også bruges til at løbe et Array igennem og erstatter altså ikke bare brugen af iteratorer, men giver også en fælles syntax for at løbe Arrays og klasser med iteratorer igennem.