# Nye features i Java 1.7/7.0

## Denne artikel beskriver nye features i Java 1.7/7.0, som blev releaset 28. juli 2011.

## Den forudsætter et vist kendskab til Java.

Skrevet den **23. jun 2013** af **arne_v** I kategorien **Programmering / Java** | ⭐⭐⭐⭐☆

Historie:
V1.0 - 17/07/2011 - original
V1.1 - 30/07/2011 - add JavaDoc, ForkJoin og faktisk release

### Kodenavn Dolphin

Java 1.7/7.0 med kodenavnet Dolphin blev releaset 28. juli 2011.

Den afløser Java 1.6/6.0 med kodenavnet Mustang (se evt. artikel http://www.eksperten.dk/guide/941 "Nye features i Java 1.6/7.0").

Jeg er gammeldags og vil fortsætte med at kalde den 1.7 selvom 7.0 efterhånden er blevet det gængse!

1.7 er blevet forsinket i flere omgange (bl.a. fordi SUN blev opkøbt af Oracle mit i forløbet) og resultatet er fire et halvt år mellem 1.6 og 1.7.

På trods af de mange år er ændringerne ikke så store.

### Indledning

Jeg vil opremse nogle af de nye features i 1.7 og illustrere med små kode eksempler.

Det er dog ikke sikkert at jeg har opdaget alle de interessante nyheder.

### Tal konstanter

Java 1.7 tillader binære konstanter og brug af understreg til at gøre tal mere læselige.

Lit16.java

```
public class Lit16 {
    public static void main(String[] args) {
        int v1 = 0x03030303;
        System.out.println(Integer.toString(v1, 2));
        int v2 = 123456789;
        System.out.println(v2);
    }
```

```
}
```

Lit17.java

```
public class Lit17 {
    public static void main(String[] args) {
        int v1 = 0b00000011000000110000001100000011;
        System.out.println(Integer.toString(v1, 2));
        int v2 = 123_456_789;
        System.out.println(v2);
    }
}
```

Jeg tvivler på at disse features vil blive brugt meget.

**switch på String**

Java 1.7 tillader switch på String.

StrSwi16.java

```
public class StrSwi16 {
    public static void test(String s) {
        if(s.equals("AA")) {
            System.out.println("called with AA");
        } else if(s.equals("BB")) {
            System.out.println("called with BB");
        } else {
            System.out.println("Called with something else");
        }
    }
    public static void main(String[] args) {
        test("BB");
    }
}
```

StrSwi17.java

```
public class StrSwi17 {
    public static void test(String s) {
        switch(s) {
            case "AA":
                System.out.println("called with AA");
                break;
```

```
            case "BB":
                System.out.println("called with BB");
                break;
            default:
                System.out.println("Called with something else");
                break;
        }
    }
    public static void main(String[] args) {
        test("BB");
    }
}
```

Denne feature eksisterer i mange andre sprog og vil formentligt blive brugt i stor stil.

**Type inferens for generiske constructorer**

Java 1.7 tillader at man bruger <> for constructor hvis Java kan gætte hvilken/hvilke typer der skal bruges.

GenCon16.java

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

public class GenCon16 {
    public static void main(String[] args) {
        List<String> list = new ArrayList<String>();
        Map<String,String> map = new HashMap<String,String>();
    }
}
```

GenCon17.java

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

public class GenCon17 {
    public static void main(String[] args) {
        List<String> list = new ArrayList<>();
        Map<String,String> map = new HashMap<>();
    }
}
```

**Multi catch**

Java 1.7 tillader at man catcher flere exceptions i en enkelt catch.

MulCat16.java

```
public class MulCat16 {
    public static void test() throws SomeException, SomeOtherException {
        throw new SomeOtherException();
    }
    public static void main(String[] args) {
        try {
            test();
        } catch(SomeException ex) {
            ex.printStackTrace();
        } catch(SomeOtherException ex) {
            ex.printStackTrace();
        }
    }
}

class SomeException extends Exception {
}

class SomeOtherException extends Exception {
}
```

MulCat17.java

```
public class MulCat17 {
    public static void test() throws SomeException, SomeOtherException {
        throw new SomeOtherException();
    }
    public static void main(String[] args) {
        try {
            test();
        } catch(SomeException|SomeOtherException ex) {
            ex.printStackTrace();
        }
    }
}

class SomeException extends Exception {
}

class SomeOtherException extends Exception {
}
```

En ganske praktisk feature som nok skal blive brugt.

**try with ressource**

Java 1.7 gør det muligt automatisk at få kaldt close uanset exception eller ej.

Præcis samme ide som using i C#/VB.NET men med en lidt anden implementation.

Lad os først prøve med noget standard Java IO kode.

TryResStd16.java

```
import java.io.FileReader;
import java.io.BufferedReader;
import java.io.IOException;

public class TryResStd16 {
    public static void main(String[] args) throws IOException {
        BufferedReader br = null;
        try {
            br = new BufferedReader(new FileReader("java17.txt"));
            String line;
            while((line = br.readLine()) != null) {
                System.out.println(line);
            }
        } finally {
            if(br != null) {
                br.close();
            }
        }
    }
}
```

TryResStd17.java

```
import java.io.FileReader;
import java.io.BufferedReader;
import java.io.IOException;

public class TryResStd17 {
    public static void main(String[] args) throws IOException {
        try (BufferedReader br = new BufferedReader(new
FileReader("java17.txt"))) {
            String line;
            while((line = br.readLine()) != null) {
                System.out.println(line);
            }
```

```
        }
    }
}
```

Man kan det samme med sin egen kode.

TryResCus17.java

```
public class TryResCus17 {
    public static void main(String[] args) throws Exception {
        try (MyRes r = new MyRes(false)) {
            r.test();
        }
        try (MyRes r = new MyRes(true)) {
            r.test();
        }
    }
}

class MyRes implements AutoCloseable {
    private boolean willthrow;
    public MyRes(boolean willthrow) {
        this.willthrow = willthrow;
        System.out.println("constuctor called");
    }
    public void test() throws Exception {
        if(willthrow) {
            throw new Exception("Ooops");
        }
    }
    public void close() {
        System.out.println("close called");
    }
}
```

Ovenstående ignorerer en del specialtilfælde:
- traditionel try og finally
- exceptions i close metoden

Nedenstående viser mere præcist hvad der sker.

TryResTricky17.java

```
public class TryResTricky17 {
    public static void main(String[] args) throws Exception {
        try (MyRes r = new MyRes()) {
            r.test();
        } catch(Exception ex) {
```

```
            System.out.println("catch of: " + ex.getMessage());
            for(Throwable t : ex.getSuppressed()) {
                System.out.println("suppressed: " + t.getMessage());
            }
        } finally {
            System.out.println("traditional finally");
        }
    }
}

class MyRes implements AutoCloseable {
    public MyRes() {
        System.out.println("constuctor called");
    }
    public void test() throws Exception {
        System.out.println("test called - will throw exception");
        throw new Exception("exception in test");
    }
    public void close() throws Exception {
        System.out.println("close called - will throw exception");
        throw new Exception("exception in close");
    }
}
```

Jeg tror at det vil tage tid inden folk vænner sig til denne feature, men efter noget tid tror jeg nok at den skal blive populær.

Og som det sidste kode viser, så skal man altså lige forstå de finere finesser først!

**IO utility metoder**

Java 1.7 har fået nogle utility metoder som gør det nemt at lave diverse trivielle fil operationer.

Svarende til hvad man kender fra Visual Basic, .NET etc..

Læse og skrive filer.

Files16.java

```
import java.io.InputStream;
import java.io.FileInputStream;
import java.io.OutputStream;
import java.io.FileOutputStream;
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.PrintWriter;
import java.io.FileWriter;
import java.io.File;
import java.io.IOException;
```

```
public class Files16 {
    public static void main(String[] args) throws IOException {
        // binary
        InputStream is = new FileInputStream("java17.txt");
        OutputStream os = new FileOutputStream("z.txt");
        byte[] buf = new byte[1000];
        int n;
        while((n = is.read(buf, 0, buf.length)) > 0) {
            os.write(buf, 0, n);
        }
        os.close();
        is.close();
        (new File("z.txt")).delete();
        // text
        BufferedReader br = new BufferedReader(new FileReader("java17.txt"));
        PrintWriter pw = new PrintWriter(new FileWriter("z.txt"));
        String line;
        while((line = br.readLine()) != null) {
            pw.println(line);
        }
        pw.close();
        br.close();
        (new File("z.txt")).delete();
    }
}
```

Files17.java

```
import java.io.IOException;
import java.nio.charset.Charset;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.nio.file.Files;

public class Files17 {
    public static void main(String[] args) throws IOException {
        Path from = Paths.get("java17.txt");
        Path to = Paths.get("z.txt");
        // binary
        Files.write(to, Files.readAllBytes(from));
        Files.delete(to);
        // text
        Files.write(to, Files.readAllLines(from,
Charset.forName("ISO-8859-1")), Charset.forName("ISO-8859-1"));
        Files.delete(to);
        // lazy
        Files.copy(from, to);
        Files.delete(to);
    }
}
```

Avancerede funktioner på fil systemet.

FS17.java

```
import java.io.IOException;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.nio.file.Files;

public class FS17 {
    public static void main(String[] args) throws IOException {
        Path f = Paths.get("java17.txt");
        System.out.println(Files.getOwner(f));
        System.out.println(Files.isReadable(f));
        System.out.println(Files.isWritable(f));
        System.out.println(Files.readAttributes(f, "*"));
        System.out.println(Files.getFileStore(f).getUnallocatedSpace() + "/" +
Files.getFileStore(f).getTotalSpace());
    }
}
```

En blanding af praktiske og længe savnede metoder som nok skal blive brugt.

**Fork join**

Java 1.6 tilføjede java.util.concurrent, men i 1.7 er der tilføjet fork join til denne pakke.

For join er et framework til at løse opgaver parallelt via divide and conquer.

FJ16.java

```
public class FJ16 {
    private int val;
    private int start;
    private int end;
    public FJ16(int val, int start, int end) {
        this.val = val;
        this.start = start;
        this.end = end;
    }
    public int countMultipla() {
        int res = 0;
        for(int i = start; i <= end; i++) {
            if(i % val == 0) {
                res++;
            }
        }
        return res;
```

```
    }
    public static void main(String[] args) {
        long t1 = System.currentTimeMillis();
        FJ16 o = new FJ16(4, 1, 1000000000);
        int n = o.countMultipla();
        long t2 = System.currentTimeMillis();
        System.out.println(n + " in " + (t2 - t1));
    }
}
```

FJ17.java

```
import java.util.concurrent.ForkJoinPool;
import java.util.concurrent.RecursiveTask;

public class FJ17 extends RecursiveTask<Integer> {
    private int val;
    private int start;
    private int end;
    public FJ17(int val, int start, int end) {
        this.val = val;
        this.start = start;
        this.end = end;
    }
    public int countMultipla() {
        int res = 0;
        for(int i = start; i <= end; i++) {
            if(i % val == 0) {
                res++;
            }
        }
        return res;
    }
    @Override
    public Integer compute() {
        int n = end - start + 1;
        if(n < 10_000_000) {
            return countMultipla();
        } else {
            FJ17 low = new FJ17(val, start, start + n/2);
            FJ17 high = new FJ17(val, start+n/2+1, end);
            // do not use:
            //invokeAll(low, high);
            //return low.join() + high.join();
            // use:
            high.fork();
            return low.compute() + high.join();
        }
    }
    public static void main(String[] args) {
        ForkJoinPool fjp = new ForkJoinPool(8);
```

```
        long t1 = System.currentTimeMillis();
        FJ17 o = new FJ17(4, 1, 1_000_000_000);
        fjp.invoke(o);
        int n = o.join();
        long t2 = System.currentTimeMillis();
        System.out.println(n + " in " + (t2 - t1));
    }
}
```

Som man kan se giver brug af fork join mere kode.

Men køre tiderne på min PC er 4626 og 1174 - altså stort set en perfekt reduktion af køre tid til 1/4 på en quad core maskine.

Bemærk at ParallelArray klassen er udskudt til Java 8, fordi lambda udtryk blev udskudt til Java 8.

Fork join skal nok blive brugt, men det bliver sjovere i Java 8, fordi Java 7 har kun leveret det grundliggende framework, mens det er Java 8 som skal spare udvikleren kode.

Det er også først med Java 8 at fork begynder at ligne .NET TPL.

Men kan man ikke vente så er her en implementation af en del af ParallelArray.

PA16.java

```
public class PA16 {
    public static void addOneToAll(int[] a) {
        for(int i = 0; i < a.length; i++) {
            a[i] = a[i] + 1;
        }
    }
    public static void main(String[] args) {
        int[] a = new int[1000];
        for(int i = 0; i < a.length; i++) {
            a[i] = i + 1;
        }
        addOneToAll(a);
        System.out.println(a[0] + " " + a[a.length - 1]);
    }
}
```

Applier.java

```
public interface Applier<T> {
    void applyToElement(T[] a, int ix);
}
```

ParallelArray.java

```
import java.util.concurrent.ForkJoinPool;
import java.util.concurrent.RecursiveAction;

public class ParallelArray<T> {
    public static class ParallelArrayApplier<T> extends RecursiveAction {
        private T[] a;
        private int start;
        private int end;
        private Applier<T> app;
        public ParallelArrayApplier(T[] a, int start, int end, Applier<T> app)
{
            this.a = a;
            this.start = start;
            this.end = end;
            this.app = app;
        }
        @Override
        public void compute() {
            int n = end - start + 1;
            if(n < 100) {
                for(int i = start; i <= end; i++) {
                    app.applyToElement(a, i);
                }
            } else {
                ParallelArrayApplier<T> low = new ParallelArrayApplier<T>(a,
start, start + n/2, app);
                ParallelArrayApplier<T> high = new ParallelArrayApplier<T>(a,
start+n/2+1, end, app);
                high.fork();
                low.compute();
                high.join();
            }
        }
    }
    private ForkJoinPool fjp;
    private T[] a;
    public ParallelArray(ForkJoinPool fjp, T[] a) {
        this.fjp = fjp;
        this.a = a;
    }
    public void apply(Applier<T> app) {
        ParallelArrayApplier<T> paa = new ParallelArrayApplier<T>(a, 0,
a.length-1, app);
        fjp.invoke(paa);
        paa.join();
    }
}
```

PA17.java

```
import java.util.concurrent.ForkJoinPool;

public class PA17 {
    private static ForkJoinPool fjp = new ForkJoinPool(8);
    public static void addOneToAll(Integer[] a) {
        ParallelArray<Integer> pa = new ParallelArray<>(fjp, a);
        pa.apply(new Applier<Integer>() {
            public void applyToElement(Integer[] a, int ix) {
                a[ix] = a[ix] + 1;
            }
        });
    }
    public static void main(String[] args) {
        Integer[] a = new Integer[1000];
        for(int i = 0; i < a.length; i++) {
            a[i] = i + 1;
        }
        addOneToAll(a);
        System.out.println(a[0] + " " + a[a.length - 1]);
    }
}
```

Jeg vil tro at koden kan forbedres, men det skulle gerne kunne sætte folk igang.

**URLClassLoader**

URLClassLoader har fået en close metode som:
- gør at classloaderen ikke længere kan loade
- closer diverse jar filer som den har holdt åben

**XML**

Den indbyggede XML support er opdateret til:
- JAXP 1.4
- JAXB 2.2a
- JAX-WS 2.2

**JDBC**

JDBC er opdateret til version 4.1.

Den største ændring er support for try with resource (se tidligere afsnit om dette)
for Connection, Statement/PreparedStatement/CallableStatement og ResultSet.

**Bedre support for dynamiske sprog**

1.7 JVM indeholder forbedret support for programmerings sprog
med dynamiske typer.

Det drejer sig bl.a. om JRuby og Jython.

Disse sprog er stigende i popularitet på Java platformen, så det er en væsentlig forbedring, men jeg vil ikke komme mere ind på den her.

**JavaDoc**

HTML designet i JavaDoc's er opdateret.

Dette er naturligvis kun en kosmetisk ændring, men det er ret synligt.

**Kommentar af T0M (nedlagt brugerprofil) d. 20. jul 2011 | 1**

Holddaop er der virkelig gået 4½ år!

De nye fil-metoder bliver et hit!
- de enkelte gange jeg har været tvunget til at lave noget i java, har jeg haft mareridt over de lede byte-array buffere..

Super artikel, tak!

**Kommentar af FelixGraversNielsen d. 01. sep 2012 | 2**

du er fantastisk håber at du vil hjælpe mig med mit spørgsmål:
http://www.eksperten.dk/spm/968280