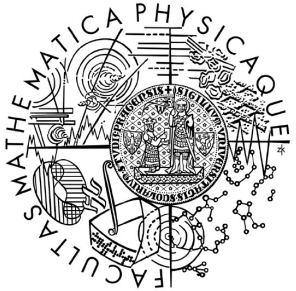
New features in Java 7

Report on *what (and when) will be (probably) in Java 7 (and 8 and …*

Department of Distributed and Dependable Systems
Faculty of Mathematics and Physics
CHARLES UNIVERSITY PRAGUE



Presentation overview

- why
  - Java is the most popular programming language
  - most of our projects use Java
- “technology radar”

Presentation overview

- why
  - Jav
  - mo
- “tech

Position Feb 2011	Position Feb 2010	Delta in Position	Programming Language	Ratings Feb 2011	Delta Feb 2010	Status
1	1	=	Java	18.482%	+1.13%	A
2	2	=	C	14.986%	-1.62%	A
3	4	↑	C++	8.187%	-1.26%	A
4	7	↑↑↑	Python	7.038%	+2.72%	A
5	3	↓↓	PHP	6.973%	-3.03%	A
6	6	=	C#	6.809%	+1.79%	A
7	5	↓↓	(Visual) Basic	4.924%	-2.13%	A
8	12	↑↑↑↑	Objective-C	2.571%	+0.79%	A
9	10	↑	JavaScript	2.558%	-0.08%	A
10	8	↓↓	Perl	1.907%	-1.69%	A
11	11	=	Ruby	1.615%	-0.82%	A
12	-	=	Assembly*	1.269%	-	A-
13	9	↓↓↓↓	Delphi	1.060%	-1.60%	A
14	19	↑↑↑↑↑	Lisp	0.956%	+0.39%	A
15	37	↑↑↑↑↑↑↑↑↑↑	NXT-G	0.849%	+0.58%	A--
16	30	↑↑↑↑↑↑↑↑↑↑	Ada	0.805%	+0.44%	A--
17	17	=	Pascal	0.735%	+0.13%	A
18	21	↑↑↑	Lua	0.714%	+0.21%	A--
19	13	↓↓↓↓↓↓	Go	0.707%	-1.07%	A--
20	32	↑↑↑↑↑↑↑↑↑↑	RPG (OS/400)	0.626%	+0.27%	A--

source: TIOBE Programming Community index
http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html

Presentation overview

- why
  - Java is the most popular programming language
  - most of our projects use Java
- “technology radar”

- presentation outline
  - Java history
  - current plans for future versions
  - overview of Java 7 included features
  - overview of Java 7 not included features (several)
    - maybe included in 8

Java history

- JDK 1.0 – 1996
- JDK 1.1 – 1997
  - inner classes
- Java 2 platform – 2000
  - JDK 1.2, 1.3 – changes in libraries only
- JDK 1.4 – 2002
  - assert
- JDK 5.0 – 2004
  - changes in language
    - generics
    - annotations
    - . . .
- JDK 6 – 2006
- JDK 7 – …

Discussed features for new Java

- discussed over the years...
  - modularization
  - anonymous methods (closures)
  - simplifications in generics implementations
  - enhanced exception handling
  - Strings in switch
  - dynamic languages support
  - …
  - …
  - updates in libraries

Current plan for Java 7

- September 2010 – Java One conference
  - Mark Reinhold: JDK 7 and Java SE 7
  - two plans
    - A – JDK 7 with all planned features – mid 2012
    - B – JDK 7 with several features – mid 2011
      JDK 8 with the rest – end 2012

- and the winner is...

Plan B

New features in Java 7

- “small” changes in syntax
  - constants
  - String in switch
  - diamond operator
  - extended try
  - multi-catch
- better support of dynamic languages

- NIO 2
- new version of JDBC
- Swing updates
- Unicode 6.0
- concurrency package updates
- …

Support of dynamic languages

- new bytecode instruction
  - invokedynamic
- new dynamic linkage mechanism
  - method handles
    - java.dyn.MethodHandle
  - bootstrap method

Constants

- binary constants
  - `0b010101`
- underscores in numeric literals
  - `1_000_000`

String in switch

```
String month;
...
switch (month) {
  case "January":
  case "February":
  ...
}
```

Diamond operator

- **<>**
  - simplified creation of generic instances
  - type is automatically inferred
  - example

```
List<String> list = new ArrayList<>();
List<List<String>> list =
                    new ArrayList<>();
List<List<List<String>>> list =
                    new ArrayList<>();
Map<String, Collection<String>> map =
                    new LinkedHashMap<>();
```

Extended try and AutoClosable

- example:

```
class Foo implements AutoClosable {
   ...
   public void close() {  …  }
}
try ( Foo f1 = new Foo(); Foo f2 = new Foo() ) {
   ...
} catch (...) {
   ...
} finaly {
   ...
}
```

- automatic call of close() on all objects from the try declaration
  - try can end regularly or with an exception
  - close() is called in opposite direction than in the declaration

Extended try and AutoClosable

- example:

```
try {
   ...
} catch (final Exception1 | Exception2 ex) {
   ...
}
```

- **why `final`**
  - if “ex” is re-thrown, the surrounding block can still catch or declare Exception1 or Exception2
    - without it, the surrounding block has to catch or declare a common super-type of Exception1 and Exception2

Changes not included in Java 7

- simplified initialization of collections
  - `List<Integer> numbers1 = [ 1, 2, 3 ];`
  - `Set<Integer> numbers2 = { 1, 2, 3 };`
  - `Map<String, String> translations =`

```
{
        "one" : "jedna",
        "two" : "dve",
        "three"  : "tri"
};
```

- access to collections via square brackets
  - `numbers1[1]`
  - `translations["one"]`

Changes not included in Java 7

- Elvis operators
  - ?:
    - binary
    - if left-hand operand is null, then the operator returns right-hand operand, else returns lef-hand operand
    - example: value = name ?: "NO-NAME"
  - ?.
    - like “dot” but only if left-hand operand is not null
    - example: person?.address?.toString();
  - ?[]
    - like [ ], i.e. access to an array or collection, but only if it is not null

Changes not included in Java 7

- Anonymous methods (closures)
  - several proposals
  - straw-man proposal (Mark Reinhold)
    - examples
      - anonymous method
        - #(int x) (x+1);
      - assigned anonymous method
        - #int(int) inc = #(int x) (x+1);
      - anonymous method with explicit return
        - #(int x, int y) { int z = foo(x, y);
if (z < 0) return x;
if (z > 0) return y;
return 0; }
      - method call
        - int y = inc.(42)
    - *proposal not accepted*

Changes not included in Java 7

- Anonymous methods (closures)
  - currently chosen solution (Lambda project)
    - can be installed to the current version of JDK 7
      - http://hg.openjdk.java.net/lambda/lambda/langtools
    - anonymous method ~ anonymous class with a single method
    - examples
      - interface Foo {
        int exec(int param);
        }
      - Foo foo = #{ int x -> x + 1 };
      - Runnable r = #{ System.out.println("Hello") };
      - interface Foo2 {
        int method(int x, int y);
        }
      - Foo2 foo2 = #{ int x, int y -> x * y };

Changes not included in Java 7

- Exception transparency (part of Lambda)
  - problem with generics
    - reasonable power at abstracting over method return types and argument types
    - useless for exceptions
  - problem illustrated

```
public interface Runnable {
    public void run();
}
```

throws nothing

```
public interface Callable<V> {
    V call() throws Exception;
}
```

throws everything

```
public interface ExceptionalCallable<V, E extends Exception> {
    V call() throws E;
}
```

useless

Changes not included in Java 7

- Exception transparency (...continued)
  - solution to the problem

```
interface Block<T, throws E> {
     public void invoke(T element) throws E;
}

interface NewCollection<T> {
    public<throws E> forEach(Block<T, throws E> block) throws E;
}
```