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
  - Java is the most popular programming language
  - most of our projects use Java
- "technology radar"
  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 month;
...
switch (month) {
    case "January":
    case "February":
        ...
    }

Diamond operator

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

```java
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:
  ```java
class Foo implements AutoClosable {
    ...
    public void close() {  ...  }
}
try ( Foo f1 = new Foo(); Foo f2 = new Foo() ) {
    ...
} catch (...) {
    ...
} finally {
    ...
}
```
- 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 left-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
• 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 };

New features in Java 7
D3S seminar, March 2, 2011
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

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

throws nothing
```

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

throws everything
```

```java
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

```java
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;
}
```