15.–18. 09. 2008
in Nürnberg

Herbstcampus

Wissenstransfer
par excellence

Abgeschlossen

Einführung in Java Closures

Michael Wiedeking

MATHEMA Software GmbH

Closures – „Definition“ …

This JSR provides support for operating on an arbitrary "block of Java code", or body, which is either a statement list, an expression, or a combination of both. We call the mechanism a closure expression. Wrapping statements or an expression in a closure expression does not change their meaning, but merely defers their execution.

Closures – „Definition”

Evaluating a closure expression produces a closure object. The closure object can later be invoked, which results in execution of the body, yielding the value of the expression (if one was present) to the invoker.
A closure expression can have parameters, which act as variables whose scope is the body. In this case the invoker of the closure object must provide compatible arguments, which become the values for the parameters.

Beispiele

```
public class SimpleClosure {

  public static void main(String[] args) {
    // function with no arguments;
    // return value is always 42
    int answer = { => 42 }.invoke();
    System.out.println(answer);
  }

}
```

Beispiele

- Closure mit einem Argument:

```
double log = { double x =>
    Math.log(x)
}.invoke(10);
```

- Closure mit einer Anweisung:

```
int half = { int x =>
    if (x % 2 != 0) {
        System.out.printf("%d is odd", x);
    }
    x / 2
}.invoke(31); // -> “31 is odd”; liefert 15
```

Beispiele

- Closure mit zwei Argumenten:

```
int sum = { int x, int y =>
    x + y
}.invoke(3, 4);  // -> 7
```

- Closure ohne Rückgabewert

```
{ char c =>
    System.out.println(c);
}.invoke('@');  // -> “@”
```

Beispiele

- Closure mit String als Rückgabewert:

```
String reversed = { String s =>
    new StringBuilder(s).reverse().toString()
}.invoke("abcd");  // -> “dcba”
```

- Closure mit Runnable als Rückgabewert

```
{ =>
    new Runnable() {
        public void run() {
            System.out.println("hi from Prague");
        }
    }
}.invoke().run();  // -> Runnable
```

Beispiele

- Closure mit lokalen Variablen

```
{ int n =>
    int m = n + 1;
    System.out.println(m * m);
}.invoke(3);  // -> “16”
```

Beispiele – statische Methoden

- Eine Methodenreferenz kann einer Funktionsvariablen zugewiesen werden:

```
{String => int} parser =
Integer#parseInt(String);
int x = parseInt.invoke("42");
```

- Mit kovarianter Rückgabe:

```
{int => Number} p = Integer#valueOf(int);
System.out.println(p.invoke(97));
```

Beispiele

- Mit kovarianten Parametern:

```
class MyClass {
    static Integer print(Object o) {
        return Integer.valueOf(o.hashCode());
    }
    public static void main(String[] args) {
        {String => Number} pp =
            MyClass#print(Object);
        System.out.println(pp.invoke("hi"));
    }
}
```

Beispiele – Instanzmethode

```
class Box {
    private int x;
    Box(int x) {
        this.x = x;
    }
    int getX() {
        return x;
    }
}
```

Beispiele

- Als echte Instantmethode

```
public class InstanceMethod {
    public static void main(String[] args) {
        Box p = new Box(10);
       {=> int} getX = p#getX();
       System.out.println(getX.invoke());
    }
}
```

Beispiele

- Oder als statische Methode mit Extra-Argument

```
{Box => int} getX = Box#getX();
Box p = new Box(10);
System.out.println(getX.invoke(p));
```

- Dabei unterliegen die Auswahlmechanismen den üblichen Verfahren

```
{Object => String} ts = Object#toString();
System.out.println(ts.invoke("hi"));
```

- Funktioniert auch mit Generics

Beispiele

```
public static void main(String[] args) {
    List<{int, int => int}> operations =
        new ArrayList<{ int, int => int }>();
    operations.add({int x, int y => x + y});
    operations.add({int x, int y => x | y});
    int[][] param = {{1, 2}, {3, 4}, {5, 6}};
    for (int[] p: param) {
        for ({int, int => int} op: operations) {
            System.out.println(op.invoke(p[0],
p[1]));
        }
    }
}
```

Beispiele

Eine Closure kann eine Closure liefern:

```
{String => {int => String}} cat = {String s =>
   {int n =>
      String r = "";
      for ( ; n > 0; n--) {
         r += s;
      }
      r
   }
};
```

Beispiele

```
{int => String} concatABC =
cat.invoke("ABC");

String result = concatABC.invoke(3);

// “ABCABCABC”
```

Currying

```
{int, int => int} plus = {int x, int y => x + y};

{int => {int => int}} anotherPlus = {int x =>
  { int y => x + y }
};

int threePlusFour =
   anotherPlus.invoke(3).invoke(4);
```

Closures – Closure Literal

- {int x, int y => x + y}
- Was ist mit
  - **this**,
  - **break**,
  - **continue** und
  - **return**?
- Neue „unchecked“ Exception UnmatchedNonlocalTransfer

Closures – Function Types

- {int, int => int} plus = {int x, int y => x + y};
- {int, String => Number **throws** IOException} o;

wird übersetzt zu

```
interface Function1<R, A2, throws E>
{
    R invoke(int x1, A2 x2) throws E;
}

Function1<? extends Number, ? super
String, ? extends IOException> o;
```

Closures – Function Types

- Funktions-Types dürfen kovariante Rückgabewerte haben
- Ein Subtyp einer Funktion muss die gleiche Anzahl Parameter haben
- Funktionen müssen kontravariante Argumente haben
- Ein Subtyp einer Funktion darf keine „checked“ Exception werfen, die nicht deklariert wurde

Kleiner Einschub für unbekannte Fremdwörter

- Arrays sind kovariant zu ihren Basis-Typen, wenn für String <= Object gilt:
  Array(String) <= Array(Object)
- Funktionen sind kontravariant zu ihren Parameter-Typen, wenn für String <= Object
  Integer g(Object)
  Integer f(String)
  ersetzen darf [f("") gilt und g("") auch]
- Allgemein sind Rückgabewerte kovariant, wenn
  Integer f(...)
  Object f(...)
  ersetzen darf

Closures – Closure Conversion

- Jedes Closure Literal kann zu einem Interface mit genau einer Methode *m* umgewandelt werden,
  - wenn gegenseitig ausschließend
    - im Literal ist kein finaler Ausdruck enthalten und der Rückgabe-Typ ist void oder java.lang.Void
    - im Literal ein finaler Ausdruck enthalten ist und es gibt eine Konvertierung von dessen Typ zum Rückgabe-Typ von *m*
    - der Rumpf des Literals kann nicht normal terminieren

Closures – Closure Conversion

- *m* hat die gleiche Anzahl Parameter wie das Literal
- für jedes korrespondierende Position der Argumente haben beide den selben Typ
- jede Ausnahme, die aus dem Rumpf des Literals geworfen werden kann, ist ein Subtyp der Ausnahmen, die von m geworfen werden können

Closure – Closure Conversion

- Implementiert das Ziel einer Closure Conversion das Interface java.lang.RestrictedFunction dann gibt es einen Übersetzungsfehler,
  - wenn das Literal ein **break**, **continue** oder **return** enthält, deren Ziele außerhalb des Literals liegen
  - wenn das Literal auf eine nicht-finale Variable zugreift, die im umliegenden Scope liegt

Closures – Closure Conversion

- Variante 1

```
interface Incrementer {
    int increment(int i);
}
Incrementer plus2 = {int i => i + 2};
```

- Variante 2

```
{int => int} plus2 = {int i => i + 2}
```

Closures – Closure Conversion

- Kombination mit dem Executor-Framework um ein Closure Literal im Hintergrund ablaufen zu lassen

```
void sayHallo(Executor executor) {
    executor.execute({=>
            System.out.println(”Hallo!”);
    });
}
```

Exception Type Parameters

```
public static <T, throws E extends Exception>
T withLock(Lock lock, {=>T throws E} block)
        throws E
{
    lock.lock();
    try {
        return block.invoke();
    } finally {
        lock.unlock();
    }
}
```

Exception Type Parameter

- Dieses Beispiel kann wie folgt benutzt werden:

```
withLock(lock, {=>
    System.out.println("Hallo!");
});
```

- Falls es nötig ist die Exception-Angabe explizit zu machen:

```
Locks.<throws IOEx|FormatEx>withLock(
    lock, {=>
        System.out.println("Hallo!");
    }
);
```

Exception Type Parameter

- Es braucht einen Typ, welcher der null für Referenzen entspricht
  - Der Typ von null ist null.**class**
  - java.lang.Null als – nicht instanziierbarer – Platzhalter
  - java.lang.Null.TYPE für null.class
- Nur damit ist dieser Typ
  - referenzierbar
  - inspizierbar

Unreachable

- Sonderfall: Nicht normal terminierender Code

```
interface NullaryFunction<T, throws E> {
   T invoke() throws E;
}

NullaryFunction<Unreachable,null> thrower = {=>
   throw new AssertionError();
};
```

Control Invocation Syntax

- Bisher

```
withLock(lock, {=>
    System.out.println("Hallo!");
});
```

- Syntaktischer Zucker!?

```
withLock(lock) {
    System.out.println("Hallo!");
}
```

Control Invocation Syntax

- Syntaktischer Zucker!?

```
withLock(lock) {
   System.out.println("Hallo!");
}
```

- statt

```
withLock(lock) {
   System.out.println("Hallo!");
};
```

Control Invocation Syntax

```
<R, T extends java.io.Closeable, throws E>
R with(T t, {T=>R throws E} block) throws E {
    try {
        return block.invoke(t);
    } finally {
        try {
            t.close();
        } catch (IOException ex) {
            ;
        }
    }
}
```

Control Invocation Syntax

- Anwendungsbeispiel

```
with(FileReader in : getReader())
    with(FileWriter out : getWriter()) {
        // code using in and out
}
```

Loop Abstractions

Vorher:

```
void test(Map<String, Integer> map) {
    for (Map.Entry<String, Integer> e :
map.entrySet()) {
        String name = e.getKey();
        Integer value = e.getValue();
        if (name.equals("end")) {
            break;
        } else if (name.startsWith("com.sun.")) {
            continue;
        }
        System.out.println(name + ":" + value);
    }
}
```

Loop Abstractions

```
<K, V, throws X>
void for eachEntry(
    Map<K,V> map,
    {K, V => void throws X} block
) throws X {
    for (Map.Entry<K,V> e: map.entrySet()) {
        block.invoke(e.getKey(), entry.getValue());
    }
}
```

Loop Abstractions

Nachher:

```
void test(Map<String,Integer> map) {
   for eachEntry(String name, Integer value:
map) {
      if (name.equals("end")) {
         break;
      } else if (name.startsWith("com.sun.")) {
         continue;
      }
      System.out.println(name + ":" + value);
   }
}
```

Beispiele

- void sayHelloInAnotherThread(Executor ex) {
  ex.execute(new Runnable() {
  public void run() {
  System.out.println("hello");
  }
  });
  }

- void sayHelloInAnotherThread(Executor ex) {
  ex.execute() {
  System.out.println("hello");
  }
  }

Beispiele

- findDivisibleBy(List<Integer> list, **final** int div)
  {
  return filter(**new** Predicate<Integer>() {
  public boolean exec(Integer arg) {
  return arg % div == 0;
  }
  });
  }

- findDivisibleBy(List<Integer> list, int div) {
  return filter({Integer x => x % div == 0},
  list);
  }
  findDivisibleBy(Arrays.asList(1, 2, 6, 9), 3) =>
  [6,9]

Vielen Dank!

Michael Wiedeking

MATHEMA Software GmbH