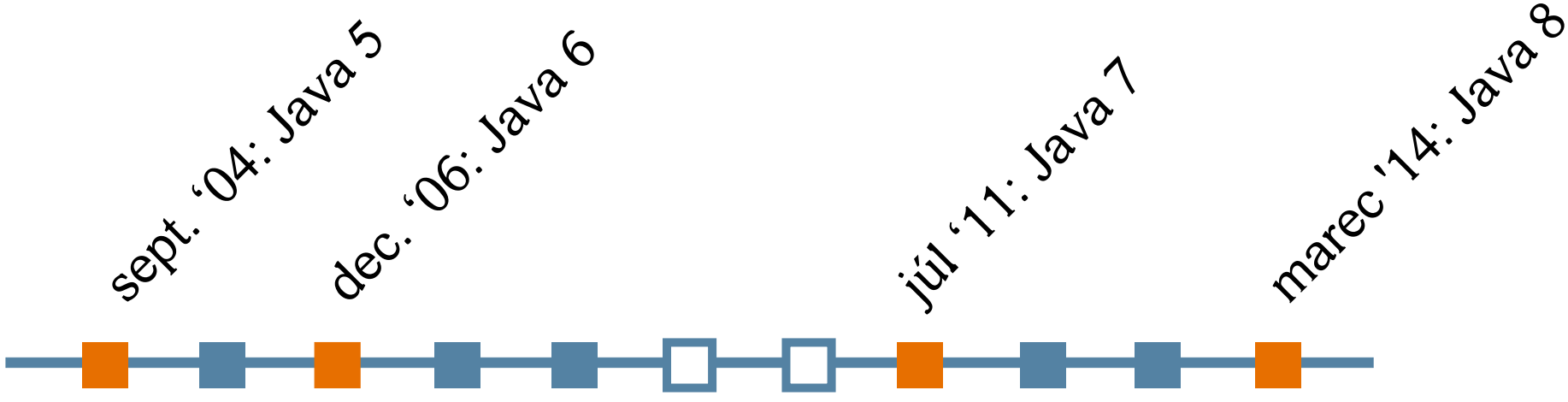
```
/** Java 8
 * @author Róbert Novotný
 * @see bezadis 09/04/14
 */
javas.stream()
  .filter(j -> j.getVersion() == 8)
```

Java ~~Sucks~~: kde sme teraz?

- dlhodobo v top 3 najpoužívanejších jazykoch // Tiobe Index 1-2
- stabilná platforma // Scala, Android
- kvitnúca komunita
- tony dokumentácie
- istota práce

História



apr. ‘09: Oracle kupuje Sun

sept ‘10: rozdelenie na Java 7 + Java 8

apr. ‘12: Java 8 oddialená o rok

Java 8: čoho sme sa dočkali?

- lambda výrazy
- polambdenie štandardnej knižnice
- defender methods
- nové API pre dátum a čas
- využiteľnejšie anotácie
- reflexia parametrov metód
- optimalizácie pre modularizáciu

Lambda výrazy

Lambda výrazy

- lambda kalkulus: formálny výpočtový model (1936)
- pracujeme s anonymnými funkciami

funkcia

= výpočet

= recept

Lambda výrazy v programovaní

- funkcia je občan prvej kategórie
- môžeme uvažovať funkcionálne
- mnoho problémov sa zrazu rieši inak
- mnoho operácií získa krajší zápis

```
public class StarýDobrýSwingovýFormulár
extends JFrame {
  public StarýDobrýSwingovýFormulár() {
    JButton button = new JButton();
    add(button);

    button.addActionListener(new ActionListener() {
      @Override
      public void actionPerformed(ActionEvent e) {
        System.out.println("Klik!");
      }
    });
  }
}
```

```
ActionListener al = new ActionListener() {
  public void actionPerformed(ActionEvent e) {
    System.out.println(”Klik!”);
  }
}
button.addActionListener(al);
```

```
ActionListener al
  = ActionEvent e -> System.out.println(”Klik!”);
button.addActionListener(al);
```

λ vs anonymné vnútorné triedy

- každá trieda implementujúca interfejs s jednou metódou sa dá zapísať lambdou

```
ActionEvent e -> System.out.println(”Klik!”);
```

parametre metódy

kód metódy

Interfejsy určené na lambdovanie

@FunctionalInterface

Inferencia parametrov

```
ActionListener al = e -> System.out.println(”Klik!”);
```

dátový typ
sa odvodí
automaticky

λ-výrazy

```
button
  .addActionListener(e->System.out.println(”Klik!”));
```

výraz s inferenciou

λ-výrazy sú výpočtom

```
ActionListener al  = e -> {
  for(int i = 0; i < 25; i++) {
    System.out.println(”I love Java”);
  }
}
```

λ-výrazy sú uzávermi (closures)

```
private int clicks = 0;

...

ActionListener al  = e-> {
  clicks++;
  System.out.println(”Number of clicks: “ + clicks);
}
```

λ-výrazy sú uzávermi (closures)

```
final JLabel label = new JLabel();
ActionListener al  = e-> {
  label.setText("Clicked!");
}
```

Lokálne premenné, ktoré chceme vidieť v lambda výraze, musia byť final!

.... pretože anonymné vnútorné triedy.

Lambdy bez parametrov

```
Runnable heartbeat = () -> {
  while(true) {
    System.out.println(”Tik!”)
  }
}

new Thread(heartbeat)
  .start();
```

Kolekcie a lambdy

```
List<String> names = Arrays.asList(...);

List<String> johnList= new LinkedList<>();
for (String name : names) {
    if(name.startsWith("john")) {
        johnList.add(name);
    }
}
```

```
List<String> johnList =
                names.stream()
                .filter(name -> name.startsWith("John"))
                .collect(Collectors.toList());
```

Streams / Prúdy

- filter: prvky spĺňajúce podmienku
- map: transformácia prvkov
- reduce: zredukovanie na 1 prvok
- sort: triedenia
- agregačné funkcie: max, min...

Každá kolekcia má prúd!

```
List<String> names = Arrays.asList(...);

List<String> capitalized = new LinkedList<>();
for (String name : names) {
    johnList.add(name.toUpperCase());
}
```

```
List<String> capitalized =
                names.stream()
                .map(name -> name.toUpperCase())
                .collect(Collectors.toList());
```

~~Pointre na funkcie~~ Method references

Metódu objektu môžeme použiť ako λ-výraz:

```
map(String name -> name.toUpperCase())

map(name -> name.toUpperCase())

map(String::toUpperCase)
```

Referencie na statické metódy

prúd čísiel mapujme na reťazce a zozbierajme do zoznamu

```
List<String> binaries =
    IntStream.range(0, 10)
    .mapToObj(Integer::toBinaryString)
    .collect(Collectors.toList());
```

public static String toBinaryString(int i)

Swing a referencie na metódy

```
public class MainForm extends JFrame {
    public MainForm() {
        JButton b = new JButton();
        add(b);

        b.addActionListener(ActionEvent e -> onButtonClick(e));
        b.addActionListener(e -> onButtonClick(e));
        b.addActionListener(this::onButtonClick);
    }

    public void onButtonClick(ActionEvent e) {
        System.out.println("Hello");
    }
}
```

Matematika

$$n! = 1 \cdot 2 \cdot \ldots \cdot n = \prod_{k=1}^{n} k$$

```
int factorial = rangeClosed(1, 5)
  .reduce((i, j) -> i * j)
  .getAsInt();
```

Paralelizmus

$$n! = 1 \cdot 2 \cdot \ldots \cdot n = \prod_{k=1}^{n} k$$

```
int factorial = rangeClosed(1, 5)
  .parallel()
  .reduce((i, j) -> i * j)
  .getAsInt();
```

redukujúca funkcia musí byť asociatívna a bezstavová

Paralelizmus

```
List<String> johns =
                names.parallelStream()
                .map(name -> name.toUpperCase())
                .collect(Collectors.toList());
```

Defender methods

Kolekcie dostali nové metódy

Interfejs java.util.Collection získal:

- stream()
- parallelStream()
- spliterator()

Všetky kolekcie sveta ich musia implementovať!

Ako dodať nové metódy do základných interfejsov a nepokaziť všetky knižnice sveta?

Defender / Default Methods

```
public interface Collection {
    ...
    default Stream<E> stream() {
        return StreamSupport.stream(spliterator(), false);
    }
}
```

Defender / Default Methods

```
public interface List<E> extends Collection<E> {
    ...
    default void sort(Comparator<? super E> c) {
        Collections.sort(this, c);
     }
}
```

Defender / Default Methods

```
List<String> names = Arrays.asList(...);

Comparator<String> byLength =
    (s1, s2) -> Integer.compare(s1.length(), s2.length());

names.sort(byLength);
```

java.time

Dátumové API cucia

- java.util.Date: céčkarský time.h struct prezlečený za objekt
- java.util.Calendar: chabý pokus o vylepšenie
- jodatime: externá knižnica s hlavou a pätou
- java.time: prevzaté a vylepšená jodatime

Časové pečiatky

```
Instant instant = Instant.now();
System.out.println(instant); // ISO-8601
```

```
2014-04-09T09:56:59.468Z
```

Lokálne dátumy

```
// miestny dátum bez časového pásma
LocalDate date = LocalDate.now();

LocalDate včera =
   LocalDate.of(2014, Month.APRIL, 8);
```

Formátovače

```
// formát podľa systémových nastavení
DateTimeFormatter formatter
  = DateTimeFormatter.ofLocalizedDate(FormatStyle.MEDIUM);

System.out.println(formatter.format(date));
```

9.4.2014

formátovače sú thread-safe

Počítanie s časom

```
LocalDate dnes = LocalDate.now();
LocalDate oTridsaťDní = dnes.plusDays(30);
```

2014-05-09

Počítanie s časom

```
LocalDate dnes = LocalDate.now();
DayOfWeek deň = dnes.getDayOfWeek();
```

WEDNESDAY

Počítanie s časom

```
YearMonth mesiac = YearMonth.now();
int početDní = mesiac.lengthOfMonth()
```

Počítanie s časom

```
LocalDate dnes = LocalDate.now();
LocalDate prvýPiatok =
   dnes.with(
     TemporalAdjusters.firstInMonth(DayOfWeek.FRIDAY));
```

2014-04-04

Lokálne a globálne dátumy a časy

```
// miestny čas bez časového pásma
LocalTime date = LocalTime.now();

// miestny dátum a čas bez časového pásma
LocalDateTime date = LocalDateTime.now();

// dátum a čas vrátane pásma
ZonedDateTime date = LocalDateTime.now();
2014-04-09T12:18:47.891+02:00[Europe/Prague]
```

Časové zóny

```
ZonedDateTime newYorkTime
  = ZonedDateTime.now(ZoneId.of("America/New_York"));
```

```
2014-04-09T07:04:51.963-04:00[America/New_York]
```

Period: uplynulá doba pre ľudí

```
LocalDate dátumNarodenia =
    LocalDate.of(1815, Month.OCTOBER, 28);
LocalDate dnes = LocalDate.now();

Period period = Period.between(dátumNarodenia, dnes);

System.out.printf("%d rokov, %d mesiacov, %d dní\n",
    period.getYears(),
    period.getMonths(),
    period.getDays());
```

198 rokov, 5 mesiacov, 12 dní

ChronoUnit: uplynulá doba

```
LocalDate dátumNarodenia =
    LocalDate.of(1815, Month.OCTOBER, 28);
LocalDate dnes = LocalDate.now();

System.out.printf("%d dní\n",
    ChronoUnit.DAYS.between(dátumNarodenia, dnes));
```

72482 dní

Duration: doba pre stroje

```
Instant teraz = Instant.now();
Instant začiatokRoka = Instant.ofEpochSecond(1388574602);
long ns = Duration.between(teraz, začiatokRoka).toNanos();
```

Čo so starým API?

- java.util.Date ~ Instant

  Date#toInstant()

  Date#fromInstant()
- java.util.GregorianCalendar ~ ZonedDateTime

  GregorianCalendar#toZonedDateTime()

  GregorianCalendar#from(ZonedDateTime)

Ďalšie novinky

Využiteľnejšie anotácie

- možnosť opakovať anotácie nad elementom

```
@RequestMapping(“/“)
@RequestMapping(“/students“)
public List<Student> list() {
...
}
```

Využiteľnejšie anotácie

- možnosť dávať anotácie na nečakané miesta
- viď JSR-308
- silnejšia typová kontrola

Map<@NonNull String, List<@Readonly Document>> files

Reflexia parametrov metód

- za behu možno zisťovať názvy parametrov metódy
- lepší autocomplete v IDE
- lepšie písanie dynamického kódu

public Student getById(String id)

Optimalizácie a modularizácia

- compact profiles

  profile1: 14MB rt.jar
- zrušenie PermGen space
- optimalizácia reprezentácie tried v pamäti (JEP-149)
- optimalizácia zdieľaného prístupu k atribútom objektu (JEP 142)
- optimalizácia HashMapy

Konkurentné programovanie

- dodané akumulátory
- prekopaná ConcurrentHashMap
- vylepšený ForkJoinPool
- prípravné práce pre STM

A zvyšok

- Nashorn: JavaScript engine + skriptovanie
- vylepšenia v bezpečnosti a kryptografii
- JavaDoc dostáva API
- apt bolo zrušené

Ako začať?

- Java 8 je už dostupná
- NetBeans 8.0: priama podpora
- IntelliJ IDEA: priama podpora od 13.1
- Eclipse:
  - 4.4 (jún 2014): oficiálne
  - 4.3 [Kepler]: možno doinštalovať feature

Kam d’alej?