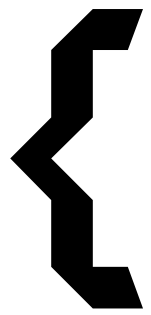


Univerzita Hradec Králové  
Fakulta informatiky a managementu

# Java 8

Pavel Janečka

prosinec 2014



# Java 8



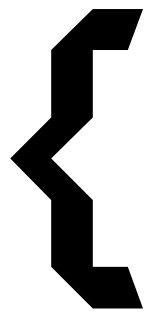
/Sorceror/GDGHK-Java8



+ Pavel Janečka



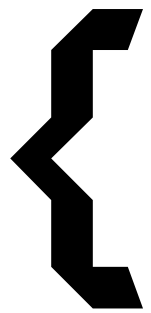
Univerzita Hradec Králové  
Fakulta informatiky a managementu



# Historie jazyka Java

- JDK 1.0 (1996) ~ private/protected
- JDK 1.1 (1997) ~ AWT, vnitřní třídy, JDBC, reflexe, JIT kompilér
- J2SE 1.2 (1998) ~ Swing, kolekce
- J2SE 1.3 (2000) ~ JNDI, JPDA (debugger)
- J2SE 1.4 (2002) ~ regulární výrazy, řetězení výjimek, podpora XML, kryptografie, Java Web Start

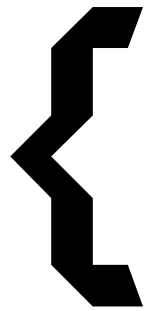




# Historie jazyka Java

- J2SE 5.0 (2004) ~ generiky, anotace, autoboxing, Varargs, for each cyklus, Swing Look and Feel, java.util.concurrent
- Java SE 6 (2006) ~ dramatické vylepšení Swing komponent, webové služby
- Java SE 7 (2011) ~ invokedynamic, NIO, Timsort, drobné úpravy syntexe
- Java SE 8 (2014) ~ ...

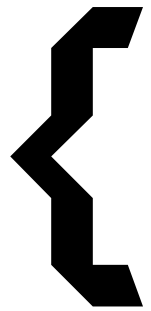




# Java 8

- Dlouhé čekání se vyplatilo
- Nejedná se o minor release
  - New Date and Time API
  - JavaScript Nashorn engine
  - Project lambda
  - Optionals
  - Streams
  - Mnoho dalších novinek



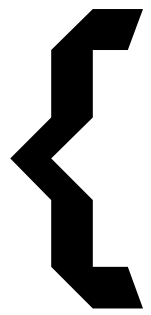


# Date and Time API

- Calendar hell vs. Joda Time

```
public static Date getMidnight(Date date) {  
    Calendar c = Calendar.getInstance();  
    c.setTime(date);  
    c.set(Calendar.HOUR_OF_DAY, 0);  
    c.set(Calendar.MINUTE, 0);  
    c.set(Calendar.SECOND, 0);  
    c.set(Calendar.MILLISECOND, 0);  
    return new Date(c.getTimeInMillis());  
}
```

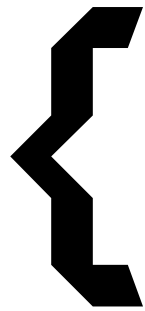




# Date and Time API

- Přepřacovaná práce s datem a časem
  - Immutable ~ thread safe





# Date and Time API

- Třídy `LocalDate`, `LocalTime`, `LocalDateTime`
  - `ZonedDateTime`

```
LocalDateTime now = LocalDateTime.now();
```

```
LocalDateTime future = now.plus(2, ChronoUnit.WEEKS)  
    .minus(17, ChronoUnit.HOURS).plusMinutes(15);
```

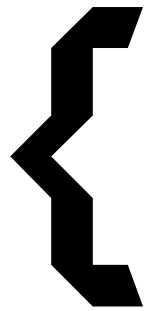
- Rozdíl mezi daty a časy

```
Period period = Period  
    .between(now.toLocalDate(), future.toLocalDate());
```

```
Duration duration = Duration  
    .between(now.toLocalTime(), future.toLocalTime());
```







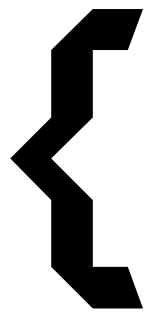
# Date and Time API

- DateTimeFormatterBuilder

```
DateTimeFormatter dtf = new DateTimeFormatterBuilder()  
    .appendLiteral("den: ")  
    .appendValue(ChronoField.DAY_OF_MONTH)  
    .appendLiteral(" ~ ")  
    .appendLiteral("mesic: ")  
    .appendValue(ChronoField.MONTH_OF_YEAR)  
    .appendLiteral(" ~ ")  
    .appendLiteral("rok: ")  
    .appendValue(ChronoField.YEAR).toFormatter();  
System.out.println(dtf.format(anotherFuture));
```

den: 5 ~ mesic: 11 ~ rok: 2014





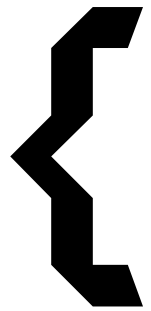
# JavaScript Nashorn engine

- Podpora skriptování aplikací pomocí ECMAScript 5.1 (později i verze 6)
  - Možnost dědit a rozšiřovat Java třídy přímo v kódu JavaScriptu
- Testovací konzole ~ `jjs`

```
ScriptEngine engine = new  
    ScriptEngineManager().getEngineByName("nashorn");  
engine.eval("print('Hello World!');");
```

<http://winterbe.com/posts/2014/04/05/java8-nashorn-tutorial/>





# Další novinky

- Třída `StringJoiner`

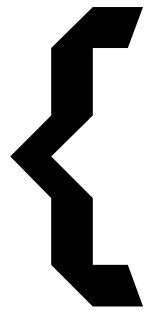
```
StringJoiner stringJoiner = new StringJoiner(", ", "{", "}");  
Stream.of(2, 7, 4, -1, 9, 12, -3, 5, -4, 0)  
    .map((n) -> Integer.toString(n))  
    .forEach(stringJoiner::add);
```

```
{2, 7, 4, -1, 9, 12, -3, 5, -4, 0}
```

```
StringJoiner anotherJoiner = new StringJoiner("|");  
anotherJoiner.add("test");  
anotherJoiner.add("test 2");  
stringJoiner.merge(anotherJoiner);
```

```
{2, 7, 4, -1, 9, 12, -3, 5, -4, 0, test|test 2}
```





# Další novinky

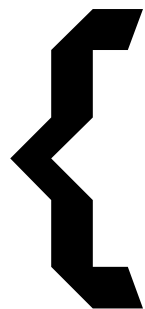
- Podpora pro Base64 kódování
- Anotace typů
  - Checker Framework

<http://types.cs.washington.edu/checker-framework/>

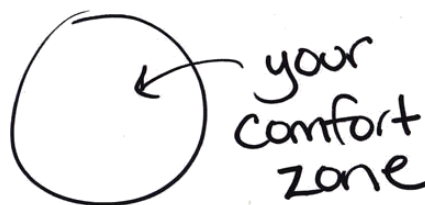
- Vícenásobné (opakující se) anotace
- Metaspace namísto PermGen

<http://java.dzone.com/articles/java-8-permgen-metaspace>



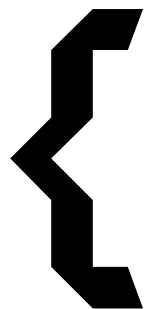


# Project lambda



- Coursera.org
  - Functional Programming Principles in Scala
  - <https://www.coursera.org/course/progfun>



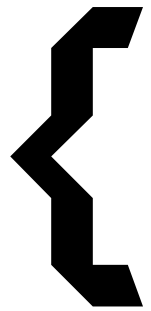


# Project



- Lambda výrazy jsou stavebním kamenem funkcionálního programování
  - NEJSOU syntaktickým cukrem anonymních vnitřních tříd
    - Mají jiný scope, First-class objekt
- <http://www.oracle.com/technetwork/articles/java/architect-lambdas-part1-2080972.html>
- Implementačně se jedná o instanční metody volané při běhu přes `invokedynamic`





# Project lambda

```
Integer[] numbers = {2, 7, 4, -1, 9, 12, -3, 5, -4, 0};
```

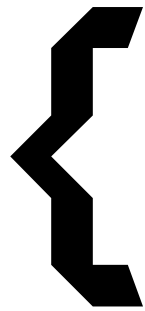
- Java 7

```
Arrays.sort(numbers, new Comparator<Integer>() {  
    @Override  
    public int compare(Integer n1, Integer n2) {  
        return n1 - n2;  
    }  
});
```

- Java 8

```
Arrays.sort(numbers, (n1, n2) -> n1 - n2);
```



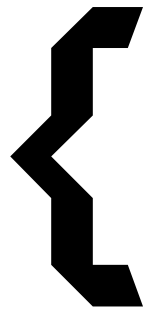


# Project lambda

- Syntaxe
  - (parametry funkce) -> tělo funkce
  - (parametry funkce) -> {  
...  
}
  - Datové typy parametrů jsou nepovinné
  - '(' a ')' jsou nepovinné pro jeden parametr
  - '{' a '}' jsou nepovinné pro jeden příkaz
  - return nepovinný pro jeden příkaz







# Project lambda

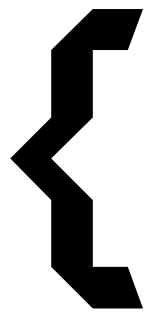
- Snadné procházení kolekcí
- Java 7

```
for (Integer n : numberList) {  
    System.out.println(n);  
}
```

- Java 8

```
numberList.forEach(n -> System.out.println(n));  
// alternative way  
numberList.forEach(System.out::println);
```



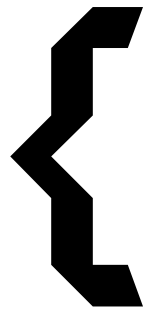


# Project lambda ~ reference

- Funkce jsou First-class objekty
  - Lze je referencovat a předávat jako parametry (funkce vyššího řádu)
- Lze referencovat
  - Statické metody
  - Instanční metody
  - Metody konkrétní instance
  - Konstruktory

<http://baddotrobot.com/blog/2014/02/18/method-references-in-java8/>





# Project lambda ~ reference

```
private class Person { ... }  
List<Person> persons = new ...
```

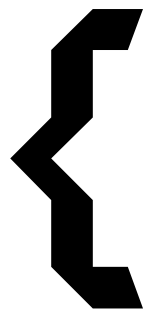
- Java 7

```
persons.sort(new Comparator<Person>() {  
    @Override  
    public int compare(Person p1, Person p2) {  
        int n = p1.getLastName().compareTo(p2.getLastName());  
        if (n == 0) {  
            return p1.getFirstName().compareTo(p2.getFirstName());  
        }  
        return n;  
    }  
});
```

- Java 8

```
persons.sort(Comparator.comparing(Person::getLastName)  
    .thenComparing(Person::getFirstName));
```

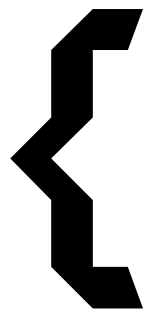




# $\lambda \sim$ **funkcionální interface**

- Interface s právě jednou abstraktní metodou
  - Runnable, Callable, Comparator, ...
- Nepovinná anotace @FunctionalInterface
- JDK obsahuje
  - Function<T,R>, Supplier<R>, Predicate<T>, Consumer<T>, BiFunction, ...
  - IntConsumer, IntFunction<R>, ...





# $\lambda$ ~ funkcionální interface

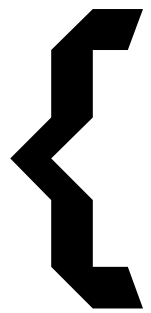
```
Integer[] numbers = {...}
```

```
private void filter(List<Integer> list,  
Predicate<Integer> condition) {  
    list.forEach(n -> {  
        if (condition.test(n))  
            System.out.println(n);  
    });  
}
```

```
// print all natural numbers  
filter(numbersList, n -> n > 0);
```

```
// print even numbers  
filter(numbersList, (n -> n % 2 == 0));
```



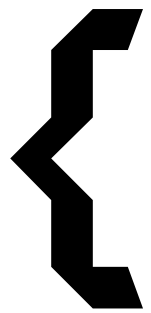


# $\lambda$ ~ defaultní metody

- Java 8 dovoluje vytvořit metody v rozhraních, které obsahují implementaci
  - Zavedeno hlavně z důvodu zpětné kompatibility streamů

```
public interface Iterable<T> {  
    Iterator<T> iterator();  
  
    default void forEach(Consumer<? super T> action) {  
        Objects.requireNonNull(action);  
        for (T t : this) {  
            action.accept(t);  
        }  
    }  
}
```





# $\lambda \sim$ statické metody rozhraní

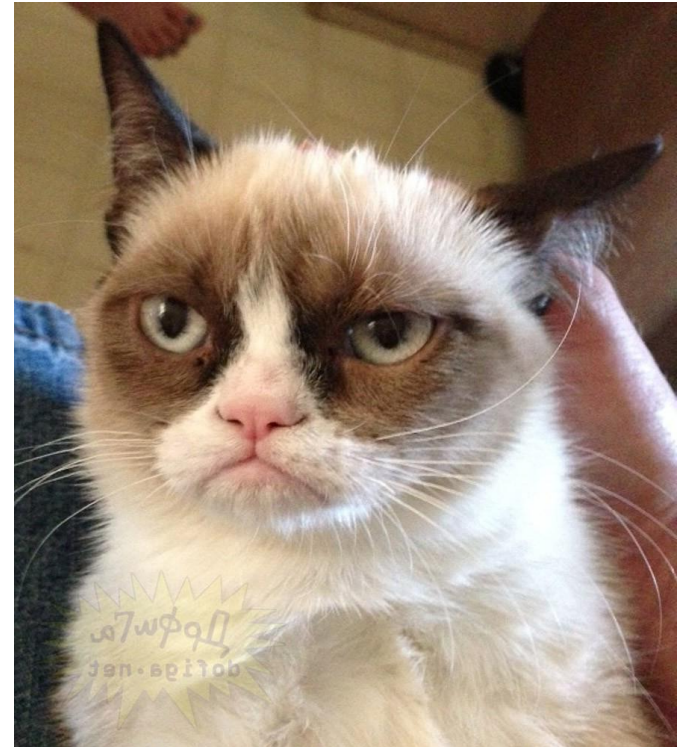
- Obdobně dovoluje Java 8 vytvořit statické metody v rozhraních
  - Opět zavedeno hlavně z důvodu streamů
  - Ideální na pomocné metody
    - Není nutné oddělovat jejich logiku do jiných tříd

```
public interface Iterable<T> {  
    Iterator<T> iterator();  
  
    static void fooUtil(String bar) {  
        ...  
    }  
}
```

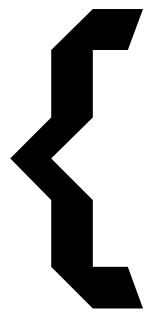


# { Optional

```
String version = "UNKNOWN";  
if (c1 != null) {  
    MB mb = c1.getMb();  
    if (mb != null) {  
        USB usb = mb.getUsb();  
        if (usb != null) {  
            ver = usb.getVer();  
        }  
    }  
}
```



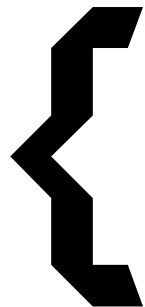




# Optional

- Nejedná se o kompletní odstranění NPE
- Nastínění způsobu návrhu a vytváření jednoznačnějšího API
- Optional<T>
  - Optional.of(...) ~ NPE pokud hodnota null
  - Optional.ofNullable(...)
  - optVal.ifPresent(Consumer<T>)
  - optVal.orElse(T defaultValue)





# Optional

- Funkce `flatMap(Function<T,R>)`
  - Aplikuje funkci na všechny prvky (vícerozměrné) kolekce tak, že výsledkem je jednorozměrná kolekce

```
String usbVersion = c1
    .flatMap(Computer::getMotherboard)
    .flatMap(Motherboard::getUsb)
    .flatMap(USB::getVersion)
    .orElse("UNKNOWN");
```



# { Streamy

- Reprezentují sekvence objektů
  - Obdobně jako rozhraní `Iterator`
- Dovolují paralelní zpracování
- Jsou lazy a immutable
  - Prvky se zpracovávají až opravdu ve chvíli kdy jsou potřeba
- Funkcionální přístup
  - `map`, `filter`, `reduce`, `limit`, `sort`, ...



# { Streamy

- Vytvoření z kolekce

```
List<Integer> numbersList =  
    Arrays.asList(2, 7, 4, -1, 9, 12, -3, 5, -4, 0);  
  
Stream numbersStream = numbersList.stream();  
  
Stream numbersParallelStream =  
    numbersList.parallelStream();
```

- Vytvoření z **Reader** instance

```
try (Stream stream = Files.lines(Paths.get(file))) {  
    stream.forEach(System.out::println);  
}
```



# { Streamy

- Generování streamů

- Seznam 100 náhodných čísel mezi 0 až 100

```
Stream.generate(Math::random).limit(100)  
.map(n -> (int)(n * 100)).collect(Collectors.toList())
```

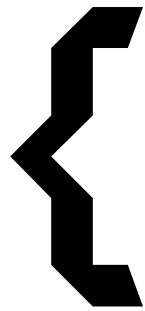
- Rozsah hodnot

```
IntStream.range(1, 11).forEach(System.out::println)
```

- Stream z ...

```
Stream.of(1, 2, 3, 4, 5)
```





# Streamy

- Procházení stromu souborů

```
try (Stream<Path> files = Files.walk(Paths.get("."))) {  
    System.out.println(  
        files.mapToLong(  
            (path) -> Files.lines(path).count()  
        ).summaryStatistics()  
    );  
}
```

- Projde soubory v aktuálním adresáři a podsložkách a vypočítá základní statistiky

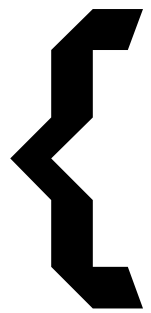
count=192, sum=5087, min=0, average=26,494792, max=2522



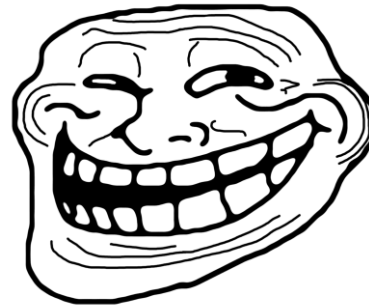
# { Streamy

- Užitečné funkce
  - filter ~ stream prvků, které splňují predikát
  - map ~ aplikuje funkci na všechny prvky
  - reduce ~ akumuluje hodnoty do jedné
  - sort ~ seřadí prvky streamu (není lazy)
  - limit ~ omezí velikost streamu
  - min/max ~ vypočítá statistiku na `IntStream`
  - peek ~ vrátí aktuální prvek bez posunu





# Java 9



- Release date 2016/17
- Project Jigsaw
  - Konkurence OSGi
- Process API
  - Vazba na procesy operačního systému
- Úprava generických typů
  - Možnost zjištění typu v runtime – yay!







# GDG Hradec Králové

```
gdghk.getEvents().stream()  
    .limit(Date.soon).peek(System.out::println)
```

- **Proč by jste měli znát Vert.x**
  - Michel Kutý
  - 22. ledna – 18.00, J6
- **Funkcionální programování a Clojure**
  - Aleš Roubíček
  - 29. ledna – 18.00, J6
- **Úvod do jazyka Scala ~ Jan Macháček**

