poročilo

KONFERENCA *JAVAONE 2011*

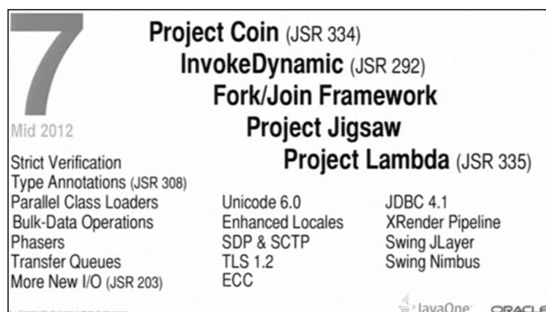
Število obiskovalcev na konferenci JavaOne 2010 se je po mnenju nekaterih glede na leto poprej podvojilo (bilo jih je okoli 9000), najverjetneje pa JavaOne ne bo več dosegla števila 12.000 obiskovalcev, kolikor jih je bilo prisotnih v času, ko se je konferenca odvijala še v centru Moscone.

Razlog za to je, da se JavaOne širi tudi regijsko. Konferenci JavaOne Russia v Moskvi (17.–18. april 2012) in JavaOne India v Hyderabadu (3.–4. maj 2012) sta se pridružili že uveljavljenima konferencama JavaOne Latin America (6.–8. december 2011) in JavaOne Tokio (4.–6. april 2012).

Tudi v letu 2011 je vzporedno potekala konferenca Oracle Open World, tako da se je JavaOne ponovno odvijala po hotelih Hilton, Union Square, Nikko ter Parc 55 (The Zone). Med hoteli se je nahajal Mason Street Café, ki je hkrati sprejel 330 ljudi. Na JavaOne 2011 je predavalo preko 500 predavateljev na 430 konferenčnih in 28 demo predavanjih.[1]

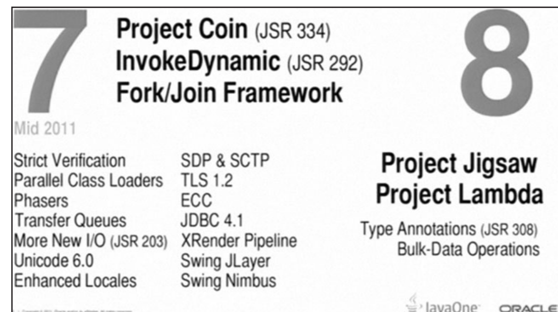
Po približno 8 mesecih, ko je Oracle prevzel Javo, je stanje takšno:

- izšel je JDK7,
- ponovno deluje JCP (The Java Community Process),
- izgradnja oblaka z Javo (java Cloud build out) je enako popularen kot pred časom izgradnja spletnih strani v Javi (java Web build out).



Java 7 bila v letu 2010 zamišljena kot nadgradnja s petimi pomembnejšimi projekti (ter kupom manjših), vendar bi izid v sredini leta 2012 pomenil že 6 let premora po izidu Jave 6. Ker tako dolgo čakanje ni prišlo v poštev, so preverili, kateri projekti se bližajo koncu in bi lahko izšli prej in nastala je nova rešitev:

PROJECT COIN (JSR 334)



Projekt Coin je zbirka sprememb v jeziku in knjižnicah Jave, narejena z namenom, da olajša vsakodnevno programiranje.

Spremembe:

- odstranitev dodatnega besedila in s tem izboljšanje berljivosti programske kode,
- spodbujanje pisanja bolj zanesljivih programov,
- dobra integracija s preteklimi in prihodnjimi spremembami,
- lažja uporaba generične kode (Diamond, Varargs warnings),
- boljša obravnava napak (Multi-catch, try-with-resources),
- doslednost in jasnost (Strings in switch, Literal improvements).

Vnos numeričnih vrednosti (literal improvements)

Dodana je bila podpora za vnos števil s podčrtajem, da bi bila berljivost boljša.

```
int phoneNumber = 555_555_1212;
long creditCardNumber = 1234_5678_9012_3456L;
long hexBytes = 0xFF_EC_DE_5E;
float monetaryAmount = 12_345_132.12;
```

Dodan pa je bil tudi vnos celih števil v binarni obliki.

```
short aShort = (short)0b1010000101000101;
int binary = 0b1011;
```

Stavek "switch" (strings in switch)

V preklopnem stavku "switch" je v Javi 7 lahko, zraven tipov char, byte, short, int, character, byte, short ter integer, po novem tudi tip string.

```
if (s.equals("foo"))        switch (s) {
   doFoo();   case "foo" : doFoo(); break;
else if (s.equals("bar"))  case "bar" :
   doBar(); break;
   doBar();   case "baz" : doBaz(); break;
else if (s.equals("baz"))  default:
   doBaz();          throw new IllegalArgume
   ntException(s);
else          }
   throw new IllegalArgumentException(s);
```

Stavek "multi-catch" ter "precise rethrow"

Sedaj lahko lovimo več različnih napak v enem stavku.

```
try {          try {
// …           // …
} catch (IOException x) {  } catch
   (IOException | SQLException x) {
   logger.log(SEVERE, "Unexpected failure",
   x);         logger.log(SEVERE,
   "Unexpected failure", x);
   throw x;          throw x;
} catch (SQLException x) { }
   logger.log(SEVERE, "Unexpected failure",
   x);
   throw x;
}
```

"Precise rethrow" nam omogoča, da nam ni treba loviti vsake napake posebej.

```
void testMethod(Future future) throws
   InterruptException, ExecutionException,
   TimeoutException {
try {
   Object result = future.get(5, SECONDS);
} catch(Throwable t) {
   cleanup();
   throw t;
}}
```

Funkcija "try-with-resources"

Je variacija funkcije "try-catch-finaly", ki omogoča, da se po uporabi viri avtomatsko zaprejo. Omogoča nam inicializacijo spremenljivke vira, ki pa mora biti tipa "AutoCloseable". Spremenljivka se zapre v avtomatsko generiranem bloku "finaly".

```
static void copy(Path src, Path dst) throws IOException {
  InputStream in = null;
  OutputStream out = null;
  try {
    in = Files.newInputStream(src);
    out = Files.newOutputStream(dst);
    byte[] buf = new byte[BUFSIZ];
    int n;
    while ((n = in.read(buf)) >= 0)
      out.write(buf, 0, n);
  } finally {
    if (in != null)
      in.close();
    if (out != null)
      out.close();
  }
}
```

JavaOne ORACLE

```
// Try-with-resources

static void copy(Path src, Path dst) throws IOException {
  try (InputStream in = Files.newInputStream(src);
       OutputStream out = Files.newOutputStream(dst))
  {
    byte[] buf = new byte[BUFSIZ];
    int n;
    while ((n = in.read(buf)) >= 0)
      out.write(buf, 0, n);
  }
}
```

JavaOne ORACLE

Ker se lahko napaka zgodi tudi pri avtomatskem zapiranju vira, je bila predstavljena nova vrsta napake "supressed exception". Tako se, če pride do več napak, ohrani prva napaka (glavna), vse druge iz avtomatsko generirane kode pa se ji dodajo kot "supressed exception".

Diamond <>

Inicializacija generičnih kolekcij je v starih verzijah Jave preveč zapletena.

```
Map<String,Map<Integer,List<String>>> map =
   new HashMap<> ();
```

ali

```
Map<String,Map<Integer,List<String>>> foo
   () {
   return new HashMap<> ();
}
```

To se imenuje operator "diamond": <>, ki odstrani deklaracijo tipa iz inicializacije.

Metoda "varargs warnings"

Pri klicanju metod "varargs" je prevajalnik generiral opozorila "unsafe". V Javi 7 je generiranje teh opozoril prestavljeno iz klicajoče kode v deklaracijo klicane metode. Dodan je bil nov anotacijski tip: SafeVarargs, ki blokira opozorila "unsafe". Vse metode iz Java 7 sedaj več ne generirajo teh opozoril.

To je do sedaj generiralo opozorila:

```
//[unchecked] unchecked generic array
   creation for varargs parameter of type
   List<String>[]
    List<List<String>>
   numbersInThreeLanguages = Arrays.
   asList(Arrays.asList("Un", "Deux",
   "Trois"),
             Arrays.asList("Uno", "Dos",
   "Tres"), Arrays.asList("One", "Two",
   "Three"));
```

Če nismo hoteli imeti opozoril, smo morali dodati:

```
@SuppressWarnings(value = "unchecked")
```

Po novem se doda anotacijski tip v klicajočo metodo:

```
@SafeVarargs
public static <T> List<T> asList(T... a) {
  return new ArrayList<T>(a);
}
```

Inicializacija kolekcij

Enostavna inicializacija kolekcij je bila prvotno mišljena za Javo 7, vendar je bila prestavljena v verzijo 8.

```
List<List<String>> monthsInTwoLanguages
= Arrays.asList(Arrays.asList("January",
"February"), Arrays.asList("Gennaio",
"Febbraio"));

List<List<String>> monthsInTwoLanguages =
   {{"January", "February"}, {"Gennaio",
   "Febbraio"}};
```

JAVA NIO

Java NIO je nov datotečni programski vmesnik, ki je nastal kot odgovor na pomanjkljivosti obstoječega vmesnika. Pomanjkljivosti obstoječega programskega vmesnika Java NIO:

- nekonsistentnost med različnimi operacijskimi okolji,
- pomanjkanje uporabnih izjem, ko pride do napak pri datotečnih operacijah,
- pomanjkanje podpore za osnovne operacije (npr. kopiranje in premikanje datotek),
- omejena podpora za simbolične povezave,
- zelo omejena podpora za datotečne atribute,
- ni funkcij, ki jih mnogo aplikacij potrebuje,
- ni možno vključiti drugih datotečnih sistemov.

Path

Osnovni razred v novem paketu je postal java.nio. file.Path, ki je nadomestil stari java.io.File. Razred je namenjen lociranju datoteke v datotečnem sistemu.

- Poti so lahko relativne ali pa absolutne.
- Ustvarimo jih lahko iz niza, URI-ja ali metode File. toPath().
- Pot sestoji iz enega ali več poimenovanih elementov, ali pa korenske komponente ter nič ali več poimenovanih elementov.
- Pot je nespremenljiva (angl. *immutable*).
- Definirane so metode za dostop do elementov poti.
- Definirane so metode za združevanje poti.

Kreiranje poti:

- `Path path = FileSystems.getDefault().getPath("C:\\myFile");`
- `Path path = Paths.get("C:\\myFile");`
- `URI u = URI.create("file:///myFile"); Path path = Paths.get(u);`
- `File f = new File("C:\\myFile"); Path path = f.toPath();`

Dostop do komponent poti:

```
//  C:\home\joe\foo
Path name = path.getFileName(); // foo
Path parent = path.getParent(); //\home\joe
Path subpath = path.subPath(0, 2);//home\joe
```

Files

Razred definira statične metode za delo z datotekami, mapami ter simbolnimi povezavami. Večina metod vzame "Path" kot vhodni parameter. Če pride do napak pri izvajanju metod, le te vračajo uporabne vhodno-izhodne izjeme.

Operacije:

```
Path path = …
```

- `Files.copy(source, target);`
- `Files.copy(source, target, REPLACE_EXISTING);`

- `Files.move(source, target);`
- `byte[] bytes = Files.readAllBytes(path);`
- `List<String> lines = Files.readAllLines(path, UTF_8);`
- ```
  //  text files
  BufferedReader reader = Files.
  newBufferedReader(path, UTF_8);
  BufferedWriter writer = Files.
  newBufferedWriter(path, ISO_8859_1);
  ```
- ```
  // input and output streams
  InputStream in = Files.
  newInputStream(path);
  OutputStream out = Files.
  newOutputStream(path);
  OutputStream out = Files.
  newOutputStream(path, CREATE, APPEND);
  ```

V paket NIO si bili dodani tudi kanali (angl. *channels*), ki predstavljajo povezave do entitet, ki so sposobne izvajati vhodno-izhodne operacije (*files, sockets*). Za operacije z datotekami skrbi razred FileChannel, ki omogoča iskanje po datoteki in zaklepanje datoteke, ter razred AsychronousFileChannel, ki zraven tega omogoča še asinhrono branje in pisanje.

Za sprehajanje po datotečni drevesni strukturi je na voljo DirectoryStream, ki za delovanje uporablja manj sistemskih virov, se prilagaja velikim mapam, ima vgrajeno filtriranje ter gladi dostopne čase do oddaljenih datotečnih sistemov.

```
Path dir = …
try (DirectoryStream<Path> stream = Files.
  newDirectoryStream(dir)) {
  for(Path entry : stream) {
  System.out.println(entry.getFileName());
}
}
```

V Javi 7 so omogočene tudi operacije nad simboličnimi povezavami, to so datoteke z referenco na drugo datoteko.

- `boolean isSymLink = Files.isSymbolicLink(path);`
- `Files.createSymbolicLink(link, target);`
- `Path target = Files.readSymbolicLink(link);`

Kljub raznolikosti izvedbe glede na platformo in datotečni sistem je bila dodana razširjena podpora za atribute datotečnega sistema. Podpirati mora vsaj osnovni vmesnik za dostop do atributov, nekatere izvedbe pa omogočajo več funkcij. Osnovni vmesnik:

```
Interface BasicFileAttributes {
  FileTime lastModifiedTime();
  FileTime lastAccessTime();
  FileTime creationTime();
  long size();
  boolean isRegularFile();
  boolean isDirectory();
  boolean isSymbolicLink();
  boolean isOther();
  Object fileKey();
}
```

Osnovne atribute pridobimo z naslednjo kodo

```
  BasicFileAttributes attrs =
  Files.readAttributes(path,
  BasicFileAttributes.class);
```

Dodan je bil nov iterator "walkFileTree", ki omogoča rekurzivno sprehajanje po drevesni strukturi. Za zagon potrebujemo začetno točko, ki jo podamo kot pot. Iterator proži "FileVisitor event" za vsako datoteko in mapo, ki jo sreča.

```
interface FileVisitor<T> {
FileVisitResult preVisitDirectory(T dir,
  BasicFileAttributes attrs);
FileVisitResult visitFile(T file,
  BasicFileAttributes attrs);
FileVisitResult vistFileFailed(T file,
  IOException ioe);
FileVisitResult postVisitDirectory(T dir,
  IOException ioe);
}
```

FileSystem

Predstavlja vmesnik do datotečnega sistema in služi kot tovarna za objekte, ki dostopajo do njega.

Vsebuje razred WatchService, ki omogoča registracijo v datotečni sistem in potem sledi vsem spremembam tega sistema. Ob spremembi registriranih poslušalcev Watchable se sproži dogodek WatchEvent.

```
WatchService watcher = FileSystems.
  getDefault().newWatchService();
Path dir = ...
WatchKey key = dir.register(watcher, ENTRY_
  CREATE, ENTRY_DELETE);
```

Spremljanje sprememb:

```
for (;;) {
WatchKey key = watcher.take();
```

```
for (WatchEvent<?> event: key.pollEvents())
   {
if (event.kind() == ENTRY_CREATE) {
Path name = (Path)event.context();
System.out.format("%s created%n", name);
}
}
key.reset();
}
```

Sedaj je omogočen razvoj in uporaba drugih izvedb datotečnega sistema. Primer je "ZIP Provider", vključen v JDK 7, ki omogoča, da vidimo vsebino datotek ZIP in JAR enako kot datotečni sistem.

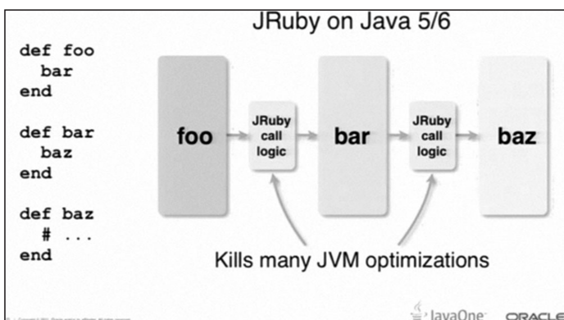
```
Path zipfile = Paths.get("foo.zip");
try (FileSystem zipfs = FileSystems.
   newFileSystem(zipfile, null)) {
   Path top = zipfs.getPath("/");
   try(DirectoryStream stream = Files.
   newDirectoryStream(top)) {
   for(Path entry : stream) {
   System.out.println(entry.getFileName());
}
}
}
```

FileStore

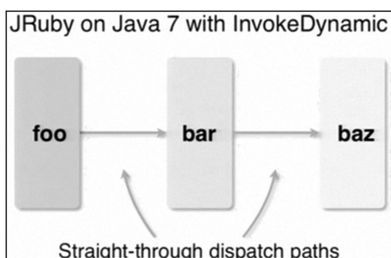
Služi kot shramba v ozadju.

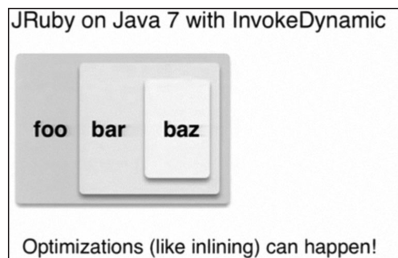
INVOKEDYNAMIC (JSR 292)

InvokeDynamic se nanaša bolj na javansko okolje kakor na Javo. Optimizacija klicanja metod vpliva na izboljšanje zmogljivosti v jJRubyju.



InvokeDynamic prinaša torej dvoje:





InvokeDynamic se je zgodilo, da je JVM prvič spremenjen za klicanje drugih jezikov v javanskem okolju in ne za Javo. Projekt Nashorn npr. je nova implementacija javascripta znotraj JVM, ki vključuje InvokeDynamic.

FORK/JOIN FRAMEWORK

Recept, kako uporabiti ogrodje Fork/Join, ki je nastalo kot posledica večjedrnih procesorjev za reševanje kompleksnih problemov, je delitev kompleksnega procesa na polovice. Te polovice se delijo v vzporedne procese, dokler problem ni tako majhen, da se procesi izvajajo zaporedno. Pri tem je pomembno, da so vsa jedra procesorja zaposlena, da je čim manj sinhronizacije in da kapacitete naraščajo linearno s številom jeder procesorja.

```
Result compute(Task t) {
    if (t.size() < SEQUENTIAL_THRESHOLD) {
        return t.computeSequentially();
    } else {
        Result left, right;
        INVOKE-IN-PARALLEL {
            left = compute(p.leftHalf());
            right = compute(p.rightHalf());
        }
        return combine(left, right);
    }
}
```

JavaOne ORACLE

Vsaka veja (angl. *fork*) vrača delni rezultat do stičišča (angl. *join point*) in ti delni rezultati se po sinhronizaciji združijo v končni rezultat. Ogrodje pri tem samo poskrbi za paralelizem in sinhronizacijo.

PROJEKT LAMBDA (JSR 335)

```
class Student {
  String name;
  int gradYear;
  double score;
}

List<Student> students = ...;

double max = Double.MIN_VALUE;
for (Student s : students) {
  if (s.gradYear == 2011)
    max = Math.max(max, s.score);
}
return max;
```

JavaOne ORACLE

Smisel izrazov lambda se dá najbolje ponazoriti z zamenjavo vektorja in zanke skozi ves seznam s podatkovno strukturo. Iskanja po podatkovni strukturi se s kopico dodatnih razredov da zapisati kot:

```
double max
  = students.filter(new Predicate<Student>() {
        public boolean eval(Student s) {
          return s.gradYear == 2011;
        }
    }).map(new Mapper<Student,Double>() {
        public Double map(Student s) {
          return s.score;
        }
    }).reduce(0.0, new Reducer<Double,Double>() {
        public Double reduce(Double max, Double score) {
          return Math.max(max, score);
        }
    });
```

Izrazi lambda gredo naprej, kodo poenostavijo tako, da kompleksni izrazi nadomestijo prej prikazane metode. Vpisati je treba le "parallel" in izvajanje se dogaja vzporedno:

```
double max
  = students.parallel()
            .filter(s -> s.gradYear == 2011) // Iterable
            .map(s -> s.score)               // Iterable
            .reduce(0.0, Math#max);          // Double
```

PROJECT JIGSAW

Gre za modularizacijo Jave, prednost je manjša velikost (angl. *download size*), krajši čas zagona in manjša poraba spomina. Modularne komponente Java je lažje uporabiti kot naravne pakete v različnih operacijskih sistemih.

Da bi uresničili celotni potencial modularizacije JDK-ja in aplikacij, mora biti tudi javansko okolje zgrajeno modularno. To bi omogočilo aplikacijam uporabo komponent JDK-ja, ki jih dejansko potrebujejo.

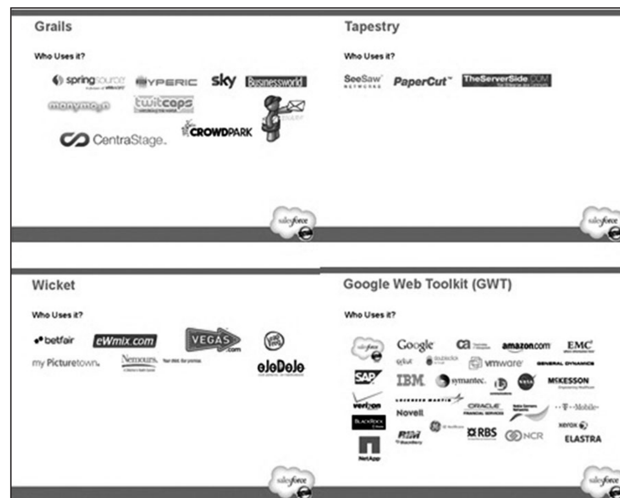
Načela načrtovanja:

- *Modularnost kot jezikovni konstrukt*.
  Najboljši način za podporo modularnemu programiranju je razširiti Javo, moduli so le ena vrsta programskih komponent.
- *Meje modulov*.
  Treba je določiti dostopnost razredov in vmesnikov in ne le preglednost. Razred, ki je zaseben glede na modul, mora biti zaseben na povsem enak način, kot je spremenljivka zasebna glede na razred.
- *Običajno zadostuje statična različica modula.*
  Večina aplikacij ne potrebuje dinamičnega dodajanja ali odstranjevanja modulov, niti ne potrebuje hkratne uporabe več različic istega modula.

JAVA WEB FRAMEWORK – PRIMERJAVA

Obstaja že več kot 110 odprtokodnih virov, ki slonijo na javanskem okolju. Kako izbrati pravega? Katera merila je treba uporabiti? Kako vedeti, da smo izbrali pravo rešitev?

Richard Pack je to temo predstavil že na Java One 2008 in ponovno je povedal, da lahko z razvojem v določenem ogrodju zaidemo v slepo ulico (nakopičena koda nam ne omogoča ponovne uporabe, ker je specifična le za določeno ogrodje). Še zmeraj je ključnega pomena hitrost razvoja: kako hitro pridemo do prototipa (krivulja učenja), koliko časa potrebujemo, da dodamo določeno storitev, ali je spreminjanje lastnosti enostavno. Ob klasični primerjavi po kriterijih: podpora AJAX, jezik (Java/JSP/OGNL …), potegni/spusti (angl. *push/pull*), zrelost, skupnost, podpora 3 Tier F/W, uporabniku prijazni URL-ji, licenca; je tokrat naredil primerjavo tudi po uporabi (dejansko se vidi, koliko je tehnologija zrela glede na to, kdo jo uporablja). Primerjava štirih najbolj popularnih:

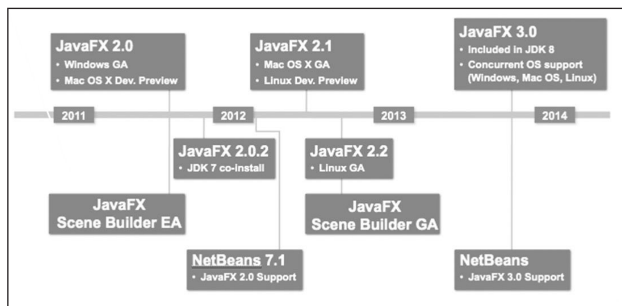


JAVAFX 2.0

Javni API JavaFX zagotavlja svobodo in prilagodljivost pri ustvarjanju bogatih odjemalcev. V okviru konference je bilo objavljeno, da od decembra 2011 teče projekt OpenJFX.

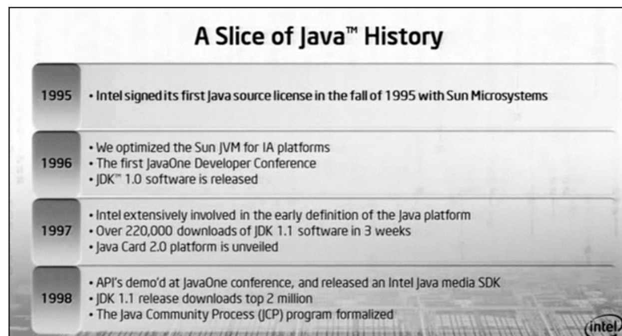
JavaFX 2 na strani uporabnika vključuje zmogljivosti okolja Java: anotacije, večnitnost, generike in razširjene knjižnice Jave. Izhodiščna točka aplikacije je Scene graph in je odgovorna za generiranje uporabniških vmesnikov in upravljanje vnosa uporabnika preko različnih vizualnih elementov, vpetih v hierarhičnem drevesu vozlišč. Vsak predmet v Scene graphu se imenuje vozlišče in ima enega starša in nič ali več otrok. API javafx.scene poenostavlja delo z bogatimi UI-ji. Grafični pogon Prism lahko deluje tako s strojnim kot programskim generatorjem (renderjem) in vključuje 3-D modeliranje.

Spletna komponenta generatorja podpira: HTML5, CSS, JavaScript, DOM in SVG, kar omogoča razvijalcem vključevanje funkcij, kot so generator vsebine HTML, podpora zgodovine, navigacija nazaj in naprej, izvajanje ukazov JavaScript in upravljanje dogodkov.

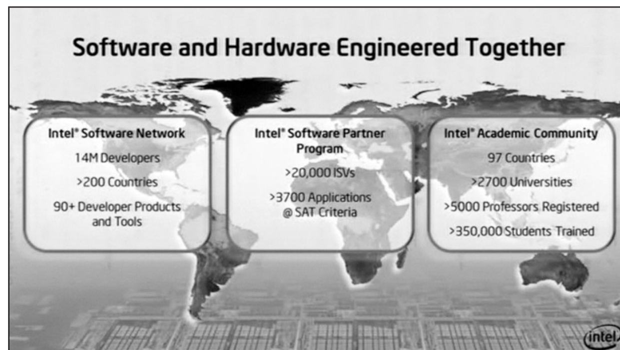


INTEL IN JAVA

"Intel se že približno 10 let ukvarja z optimizacijo Jave in če pogledamo zgodovino, so v 1996 ustvarili ekipo, ki se je začela ukvarjati z optimizacijo JVM-ja.



Po 10 letih razvoja se danes Java nahaja na 115 mio TV-jev in 1 miljardi pametnih telefonov. Java ima ključno vlogo pri zagotavljanju in razvijanju izvajalnih okolij na teh napravah in kar je še bolj pomembno, kar 97 % podjetij uporablja Javo. To je mogoče, ker je Java v 2010 in tudi v 2011 najbolj zastopan programski jezik.



Opombe

1 http://www.oracle.com/us/corporate/press/mediakits/javaone-2011-fun-facts-511910.pdf
2 http://stronglytypedblog.blogspot.com/2011/02/project-coin-examples-with-jdk-7.html
3 https://oracleus.wingateweb.com/published/oracleus2011/sessions/23360/S23360%20_1546990.pdf
4 https://oracleus.wingateweb.com/published/oracleus2011/sessions/22641/S22641_2634750.pdf
5 http://home.izum.si/COBISS/OZ/2011_1-2/html/clanek_05.html#d0e187
6 https://oracleus.wingateweb.com/published/oracleus2011/sessions/23424/S23424_138066.pdf

Stašo Vobič, Andrej Barovič Karpov