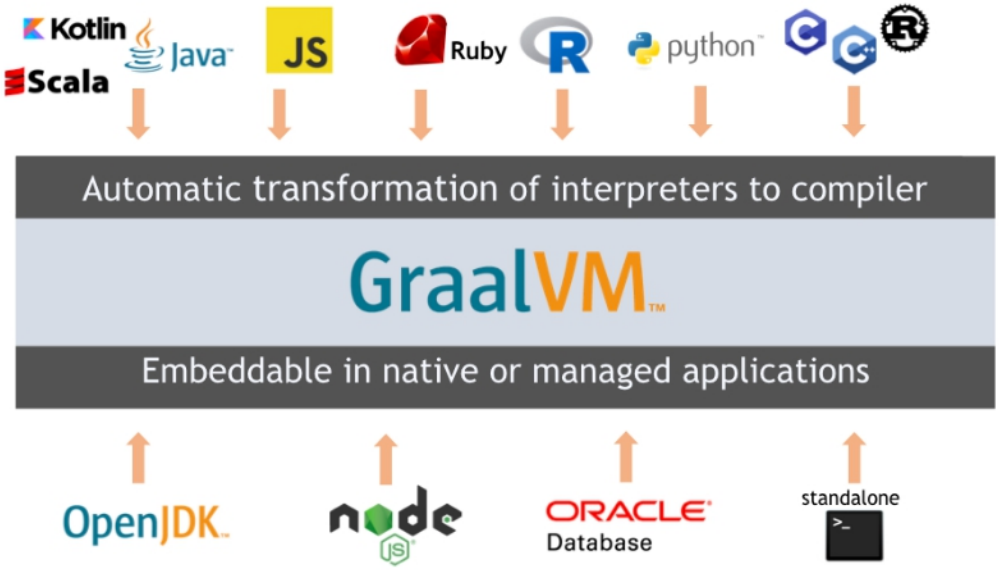
Paradigme de Programare - Laborator 2

Utilizarea limbajului PolyGlot cu GraalVM

GraalVM suportă, după cum am discutat, apeluri de instrucțiuni aparținând mai multor limbaje. În exemplul de mai jos avem o aplicație schelet (template) care, utilizând un tablou de cuvinte, va calcula o sumă de control și apoi afișa cuvintele împreună cu suma de control calculată pentru acestea. Acest lucru se poate face în orice limbaj, însă aici întâlnim utilizarea simultană a patru limbaje diferite.



Sursa: https://www.graalvm.org/docs/introduction/

Instalarea și configurarea GraalVM

Descărcați ultima versiune de GraalVM Community Edition de aici:
https://github.com/graalvm/graalvm-ce-builds/releases

graalvm-ce-java11-linux-aarch64-22.0.0.2.tar.gz	399 MB
graalvm-ce-java11-linux-aarch64-22.0.0.2.tar.gz.sha256	64 Bytes
graalvm-ce-java11-linux-amd64-22.0.0.2.tar.gz	405 MB
graalvm-ce-java11-linux-amd64-22.0.0.2.tar.gz.sha256	64 Bytes
graalvm-ce-java11-windows-amd64-22.0.0.2.zip	322 MB
graalvm-ce-java11-windows-amd64-22.0.0.2.zip.sha256	64 Bytes

Dezarhivați conținutul folosind un arhivator grafic, sau utilizând comanda:

```
tar xvf graalvm-ce-java11-linux-amd64-22.0.0.2.tar.gz
```

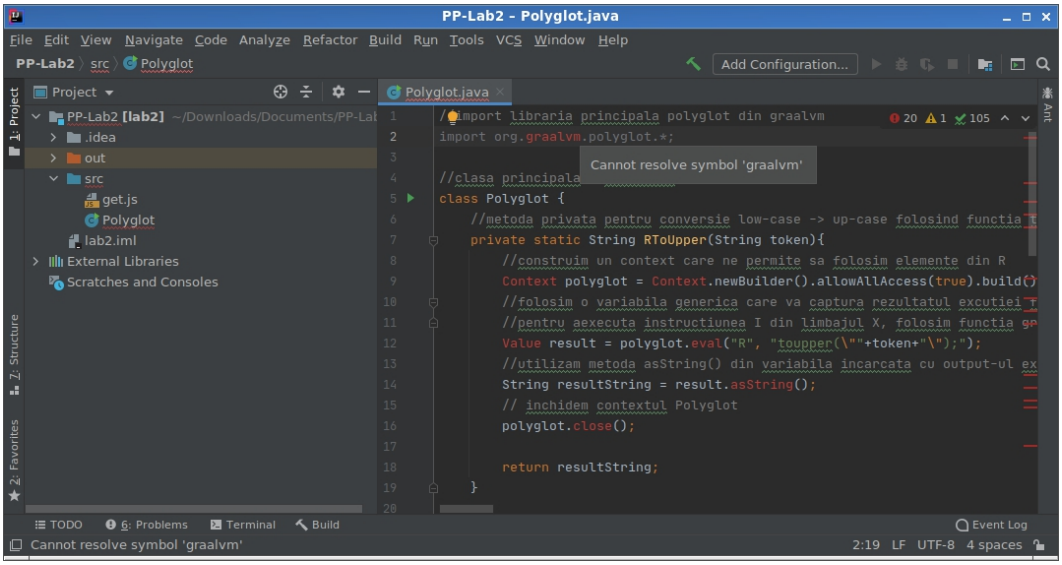
Înlocuiți versiunea marcată cu roșu corespunzător cu cea pe care ați descărcat-o.

Binarele GraalVM sunt disponibile în folder-ul **graalvm-ce-java11-22.0.0.2/bin**. Presupunând că aveți un terminal deschis în folder-ul **graalvm-ce-java11-22.0.0.2**, testați astfel:

```
./bin/java -version
```

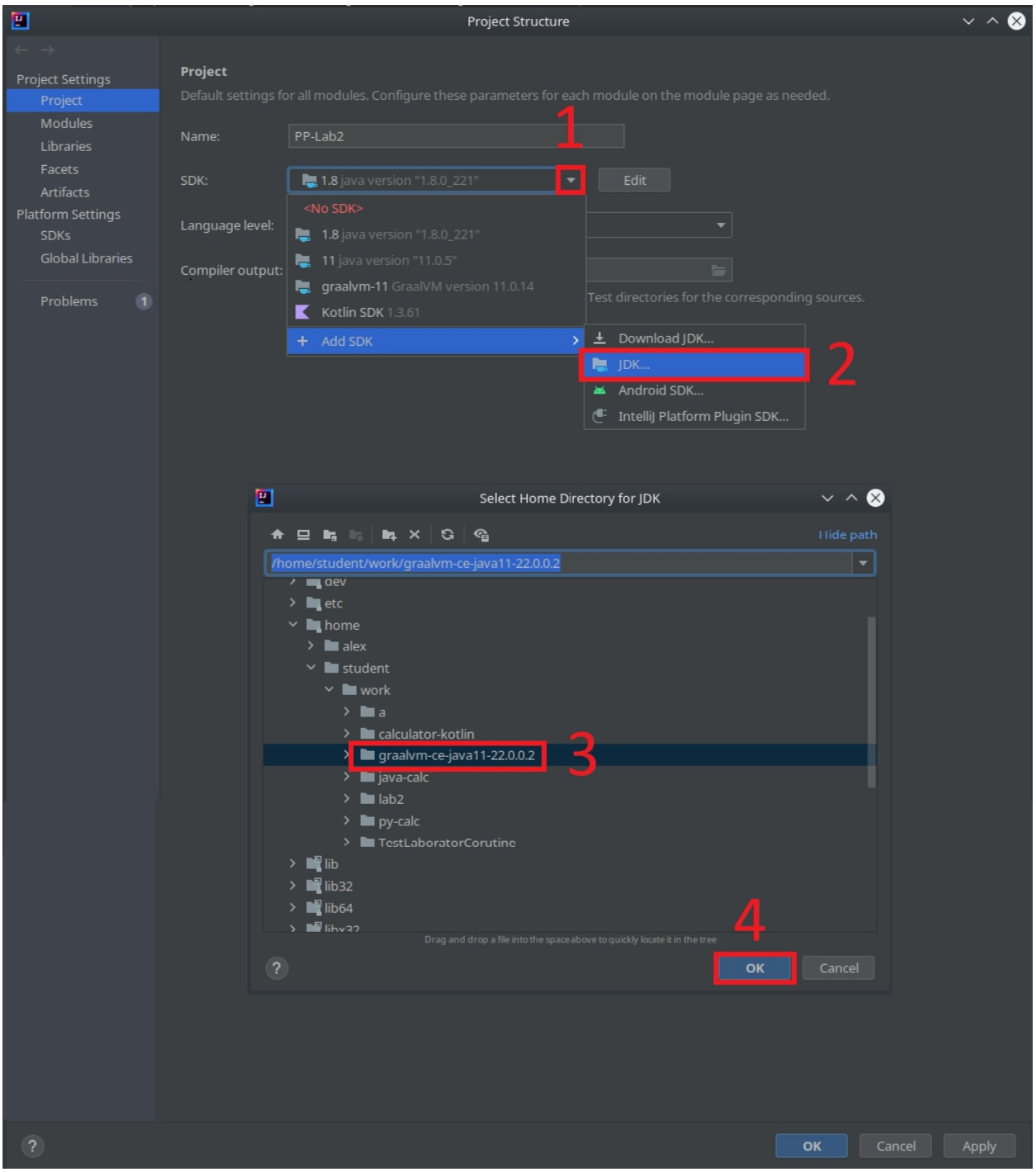
```
openjdk version "11.0.14" 2022-01-18
OpenJDK Runtime Environment GraalVM CE 22.0.0.2 (build 11.0.14+9-jvmci-22.0-b05)
OpenJDK 64-Bit Server VM GraalVM CE 22.0.0.2 (build 11.0.14+9-jvmci-22.0-b05, mixed mode, sharing)
```

Deschideți proiectul care conține codul exemplu din laborator folosind IntelliJ. Veți observa că IntelliJ nu găsește biblioteca GraalVM, de unde se importă Polyglot:



Configurați proiectul să folosească SDK-ul GraalVM pentru compilare și execuție: **File → Project Structure** → click pe **Project** în partea stângă → verificaţi să fie selectat **GraalVM** în lista din secţiunea **Project SDK**.

Dacă nu există, adăugaţi SDK-ul astfel: click pe **Add SDK** în dreptul acelei liste → selectaţi JDK.



IntelliJ va adăuga noul SDK în listă, dar sub un nume care nu e prea sugestiv. Daţi click pe butonul **Edit** din dreptul listei de SDK-uri şi redenumiţi-l sub numele „**GraalVM**”.

Instalarea limbajelor suplimentare pentru GraalVM

Dacă încercaţi să executaţi aplicaţia şi primiţi o eroare de acest tip:

```
Exception in thread "main" java.lang.IllegalArgumentException: A language with id 'R' is not installed. Installed languages are: [js, llvm].
    at com.oracle.truffle.polyglot.PolyglotEngineImpl.requirePublicLanguage(PolyglotEngineImpl.java:853)
    at com.oracle.truffle.polyglot.PolyglotContextImpl.requirePublicLanguage(PolyglotContextImpl.java:842)
    at com.oracle.truffle.polyglot.PolyglotContextImpl.eval(PolyglotContextImpl.java:813)
    at org.graalvm.polyglot.Context.eval(Context.java:344)
    at org.graalvm.polyglot.Context.eval(Context.java:370)
    at Polyglot.RToUpper(Polyglot.java:12)
    at Polyglot.main(Polyglot.java:37)
```

înseamnă că nu aveţi limbajul indicat de eroare instalat în SDK-ul GraalVM.

Pentru a instala un limbaj nou, deschideți un terminal în folder-ul GraalVM (în laborator, /home/student/opt/graalvm-ce-java11-22.0.0.2) și executați comanda:

```
./bin/gu install <NumeLimbaj>
```

Pentru acest laborator, veți avea nevoie de R și Python instalate suplimentar:

```
./bin/gu install R
./bin/gu install Python
```

```
Downloading: Release index file from oca.opensource.oracle.com
Downloading: Component catalog for GraalVM Enterprise Edition 22.0.0.1 on jdk11 from oca.opensource.oracle.com
Downloading: Component catalog for GraalVM Enterprise Edition 22.0.0 on jdk11 from oca.opensource.oracle.com
Downloading: Component catalog from www.graalvm.org
Processing Component: FastR
Processing Component: LLVM.org toolchain
```

Aplicație demonstrativă

1. Aplicația principală este scrisă în Java.

```
28      //functia MAIN
29      public static void main(String[] args) {
30          //construim un context pentru evaluare elemente JS
31          Context polyglot = Context.create();
32          //construim un array de string-uri, folosind cuvinte din pagina web:  https://chrisseaton.com/truffleruby/t
33          Value array = polyglot.eval( languageId: "js",  source: "[\"If\",\"we\",\"run\",\"the\",\"java\",\"command\",\
34          //pentru fiecare cuvant, convertim la upcase folosind R si calculam suma de control folosind PYTHON
35          for (int i = 0; i < array.getArraySize();i++){
36              String element = array.getArrayElement(i).asString();
37              String upper = RToUpper(element);
38              int crc = SumCRC(upper);
39              System.out.println(upper + " -> " + crc);
40          }
41      }
42  }
```

2. Tabloul de cuvinte este definit utilizând JavaScript.

```
32          //construim un array de string-uri, folosind cuvinte din pagina web:  https://chrisseaton.com/truffleruby/tenthin
33          Value array = polyglot.eval( languageId: "js",  source: "[\"If\",\"we\",\"run\",\"the\",\"java\",\"command\",\"incl
```

3. Înainte de calculul sumei de control, cuvintele sunt transformate cu litere mari (upper case) folosind R.

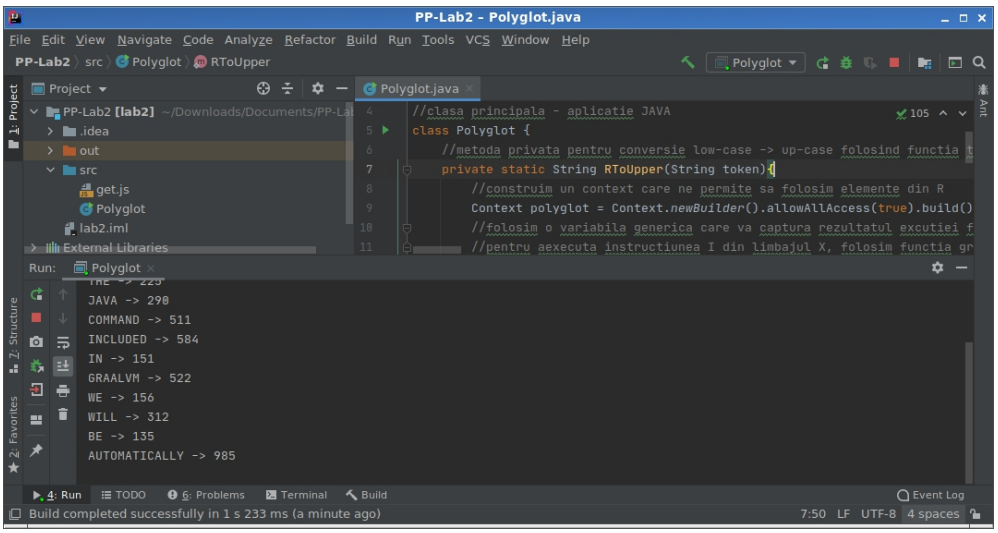
```
7       private static String RToUpper(String token){
8           //construim un context care ne permite sa folosim elemente din R
9           Context polyglot = Context.newBuilder().allowAllAccess(true).build();
10          //folosim o variabila generica care va captura rezultatul executiei functiei R, toupper(String)
11          //pentru aexecuta instructiunea I din limbajul X, folosim functia graalvm polyglot.eval("X", "I");
12          Value result = polyglot.eval( languageId: "R",  source: "toupper(\""+token+"\");");
13          //utilizam metoda asString() din variabila incarcata cu output-ul executiei pentru a mapa valoarea generica la un String
14          return result.asString();
15      }
```

4. Calculul sumei de control se realizează în Python.

```
17      //metoda privata pentru evaluarea unei sume de control simple a literelor unui text ASCII, folosind PYTHON
18      private static int SumCRC(String token){
19          //construim un context care ne permite sa folosim elemente din PYTHON
20          Context polyglot = Context.newBuilder().allowAllAccess(true).build();
21          //folosim o variabila generica care va captura rezultatul executiei functiei PYTHON, sum()
22          //avem voie sa inlocuim anumite elemente din scriptul pe care il construim spre evaluare, aici token provine din JAVA, dar va fi
23          Value result = polyglot.eval( languageId: "python",  source: "sum(ord(ch) for ch in '" + token + "')");
24          //utilizam metoda asInt() din variabila incarcata cu output-ul executiei, pentru a mapa valoarea generica la un Int
25          return result.asInt();
26      }
```

Atenţie: dacă executaţi aplicaţia din IntelliJ în laborator, modificaţi vectorul array de pe linia 41 pentru a nu conţine mai mult de 5 cuvinte. Altfel, staţiile din laborator vor rămâne fără memorie RAM în timpul execuţiei.

Aplicația în execuție:



Compilarea și execuția aplicației GraalVM din linie de comandă

Dacă executați aplicația din terminal, puteți utiliza mai multe cuvinte în vectorul **array**. Deschideți un terminal în folder-ul care conține fișierul sursă *Polyglot.java* din proiectul aplicației demonstrative, apoi introduceți următoarea comandă pentru a compila programul folosind GraalVM SDK:

```
/home/student/opt/graalvm-ce-java11-22.0.0.2/bin/javac Polyglot.java
# iar pentru a executa aplicatia:
/home/student/opt/graalvm-ce-java11-22.0.0.2/bin/java Polyglot
```

Instrumente de depanare și monitorizare

Cod javascript pentru calcularea primelor 5000 de numere prime:

```
class AcceptFilter {
   accept(n) {
      return true
   }
}

class DivisibleByFilter {
   constructor(number, next) {
      this.number = number;
      this.next = next;
   }

   accept(n) {
      var filter = this;
      while (filter != null) {
         if (n % filter.number === 0) {
            return false;
         }
         filter = filter.next;
      }
      return true;
   }
}
```

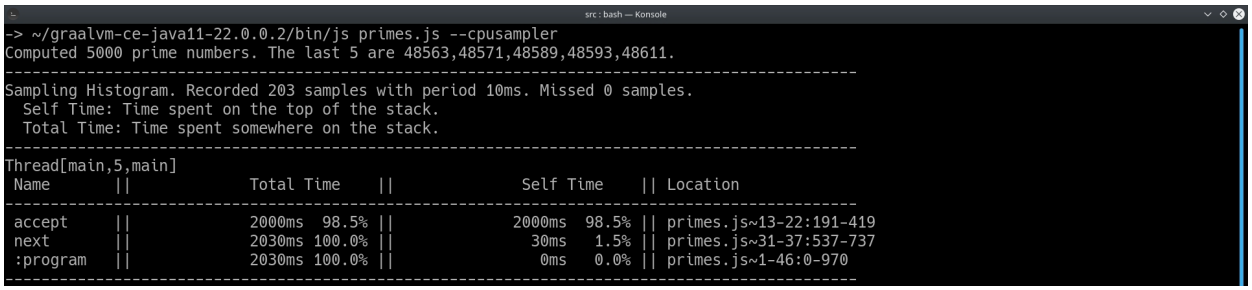
```
class Primes {
   constructor() {
      this.number = 2;
      this.filter = new AcceptFilter();
   }

   next() {
      while (!this.filter.accept(this.number)) {
         this.number++;
      }
      this.filter = new DivisibleByFilter(this.number, this.filter);
      return this.number;
   }
}

var primes = new Primes();
var primesArray = [];
for (let i = 0; i < 5000; i++) {
   primesArray.push(primes.next());
}
console.log(`Computed ${primesArray.length} prime numbers. ` +
`The last 5 are ${primesArray.slice(-5)}.`);
```

Pentru a executa codul de mai sus, deschideți un terminal în directorul cu fișierul javascript și executați comanda de mai jos, modificând calea până la graalvm corespunzător:

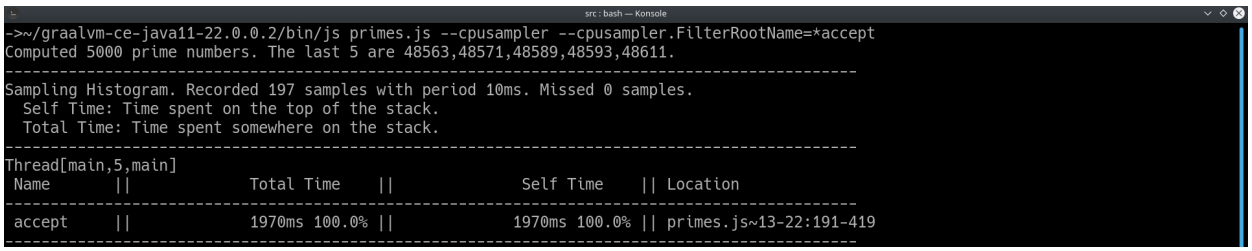
```
~/graalvm-ce-java11-22.0.0.2/bin/js primes.js
```

În funcție de parametrii pe care îi adăugați la comanda de mai sus, veți vedea detalii cu privire la:

- utilizarea CPU-ului, defalcată pe funcții (--cpusampler)



- utilizarea CPU-ului cu filtru pe denumirile funcțiilor (--cpusampler --cpusampler.FilterRootName=*accept). În acest caz, toate funcțiile care se termină cu „accept”.



- numărul de executări ale unor instrucțiuni dintr-o funcție (--cputracer --cputracer.TraceStatements --cputracer.FilterRootName=*accept)

```
src : bash — Konsole
->~/graalvm-ce-java11-22.0.0.2/bin/js primes.js --cputracer --cputracer.TraceStatements --cputracer.FilterRootName=*accept
Computed 5000 prime numbers. The last 5 are 48563,48571,48589,48593,48611.
-------------------------------------------------------------------------------------
Tracing Histogram. Counted a total of 468336895 element executions.
  Total Count: Number of times the element was executed and percentage of total executions.
  Interpreted Count: Number of times the element was interpreted and percentage of total executions of this element.
  Compiled Count: Number of times the compiled element was executed and percentage of total executions of this element.
-------------------------------------------------------------------------------------
 Name     |          Total Count |    Interpreted Count |       Compiled Count | Location
-------------------------------------------------------------------------------------
 accept   |     234117338  50.0% |         69512   0.0% |     234047826 100.0% | primes.js~15:245-258
 accept   |     117053670  25.0% |         34640   0.0% |     117019030 100.0% | primes.js~16-18:275-348
 accept   |     117005061  25.0% |         33995   0.0% |     116971066 100.0% | primes.js~19:362-381
 accept   |         53608   0.0% |           761   1.4% |         52847  98.6% | primes.js~14:211-227
 accept   |         53608   0.0% |           761   1.4% |         52847  98.6% | primes.js~13-22:191-419
 accept   |         48609   0.0% |           645   1.3% |         47964  98.7% | primes.js~17:322-334
 accept   |          4999   0.0% |           116   2.3% |          4883  97.7% | primes.js~21:402-413
 accept   |             1   0.0% |             1 100.0% |             0   0.0% | primes.js~3:45-55
 accept   |             1   0.0% |             1 100.0% |             0   0.0% | primes.js~2-4:25-61
-------------------------------------------------------------------------------------
```

- alocările de memorie (--experimental-options --memtracer)

```
src : bash — Konsole
->~/graalvm-ce-java11-22.0.0.2/bin/js primes.js --experimental-options --memtracer
Computed 5000 prime numbers. The last 5 are 48563,48571,48589,48593,48611.
-----------------------------------------------------------
 Location Histogram with Allocation Counts. Recorded a total of 5007 allocations.
   Total Count: Number of allocations during the execution of this element.
   Self Count: Number of allocations in this element alone (excluding sub calls).
-----------------------------------------------------------
 Name     |      Self Count |     Total Count | Location
-----------------------------------------------------------
 next     |     5000  99.9% |     5000  99.9% | primes.js~31-37:537-737
 :program |        6   0.1% |     5007 100.0% | primes.js~1-46:0-970
 Primes   |        1   0.0% |        1   0.0% | primes.js~25-38:424-739
-----------------------------------------------------------
```

De asemenea, GraalVM oferă posibilitatea de depanare prin atașarea unui debugger precum Chrome Developer Tools.

Pentru mai multe detalii, vezi:

- https://www.graalvm.org/tools/profiling/
- https://www.graalvm.org/tools/chrome-debugger/
- https://www.graalvm.org/tools/visualvm/

Aplicaţii şi teme

Aplicaţii de laborator:

1. Schimbați algoritmul de calcul al sumei de control, scris în **Python**, folosind o formulă polinomială de cel mult rang 5.
2. Pentru calculul sumei de control, să se elimine primul și ultimul caracter folosind funcții substring.
3. Să se implementeze o aplicație Polyglot într-un limbaj la alegere (suportat de GraalVM), care să conțină trei funcții:
    - o funcție **python** care să genereze aleatoriu o listă de 20 de numere întregi;
    - o funcție **javascript** care va afișa lista de numere întregi;
    - o funcție **R** care să sorteze lista respectivă, să elimine primele și ultimele 20% dintre elemente și să calculeze media aritmetică.

   Rezultatul final va fi afișat la consolă. Se poate pleca de la scheletul aplicației exemplu.

Teme pe acasă:

1. Modificați aplicația de laborator (după ce s-au implementat cerințele acestuia) astfel încât să afișeze toate cuvintele cu aceeași sumă de control.
2. Să se implementeze o regresie liniară utilizând limbajul **R** pe parte statistică, cu posibilitatea introducerii setului de date din python/java/orice alt limbaj cu care sunteți familiari (și este suportat de GraalVM). Rezultatul plotării (imaginea) va fi salvat pe disk și va fi deschis apelând o funcție sistem. Întrucât GraalVM nu oferă încă suport pentru majoritatea bibliotecilor grafice, se va folosi biblioteca ***lattice*** pentru plotare. Setul de date de intrare va fi introdus într-un fișier numit *dataset.txt*, fiind citit de aplicația master și trimis ca parametru funcției de regresie liniară. De asemenea, se vor citi de la tastatură numele fișierului de output (numele imaginii), calea unde se dorește să fie salvat și culorile de plotare.
3. Să se implementeze o aplicație Polyglot într-un limbaj la alegere (suportat de GraalVM) care să citească prin intermediul unei funcții **python** două numere naturale de la tastatură: numărul de aruncări ale unei monede și un alt număr x, unde $1 \leq x \leq$ numărul de aruncări. Aplicația va utiliza capabilitățile statistice ale limbajului **R** pentru a calcula distribuția binomială corespunzătoare aruncării unei monede și va afișa probabilitatea de a obține de cel mult x ori pajură din numărul total de aruncări.