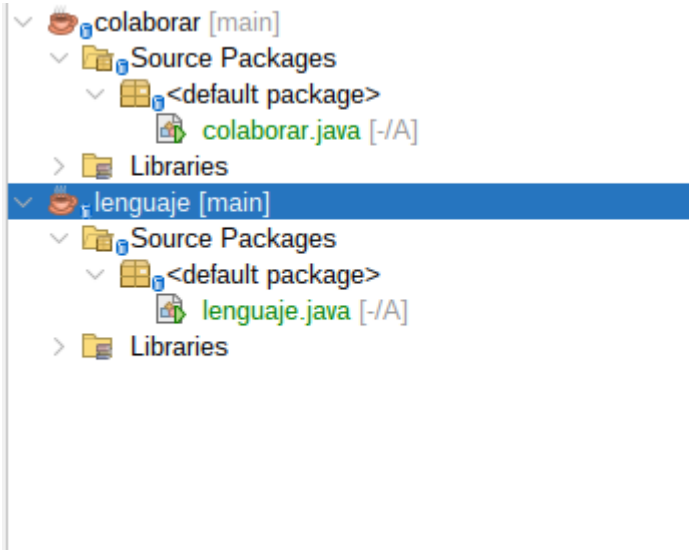
¿Cómo se hace?

Estructura de los proyectos:



El código de los proyectos se puede ver en lenguaje/src/lenguaje.java y colaborar/src/colaborar.java

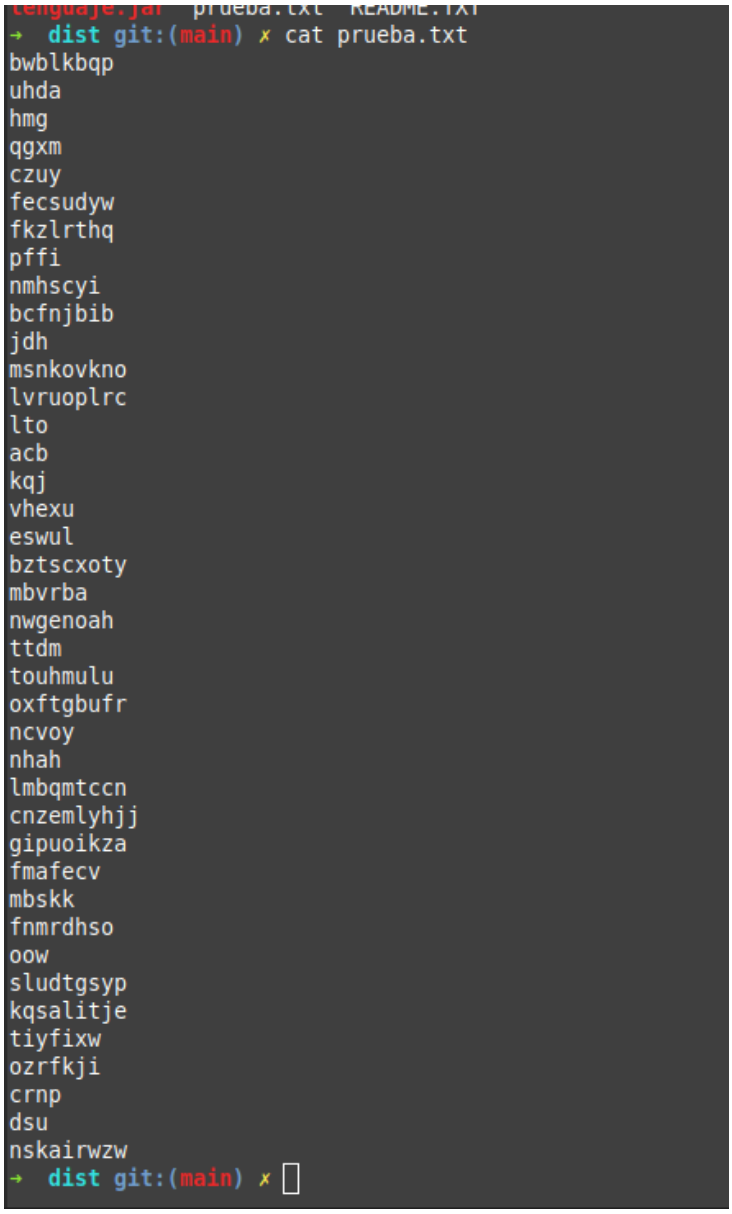
- Después de escribir el código, hemos generado un jar y generado javadocs (botón derecho en el proyecto>clean and build o botón derecho>generar javadoc, y ejecutado en terminal de la siguiente manera:

En el caso de lenguaje.java es importante hacerlo de esta manera porque sin pasarle argumentos no deja hacer la build correctamente.

- Vamos a la terminal y comprobamos que funciona el .jar

```
→  lenguaje git:(main) ✗ cd dist
→  dist git:(main) ✗ ls
lenguaje.jar  README.TXT
→  dist git:(main) ✗ java -jar lenguaje.jar 40 prueba.txt
Proceso completado: 40 palabras escritas en prueba.txt
→  dist git:(main) ✗
```

Y en prueba.txt



Se han generado nuestras palabras sin sentido

Con sus correspondientes javadoc(siento que esté en alemán, es que tengo el lenguaje del sistema en este idioma porque me voy a presentar a un examen de certificación. Klassen=Clase, Beschreibung= Descripción)

Klassen

Klasse	Beschreibung
lenguaje	Genera palabras aleatorias y las guarda en un fichero.

Pasamos a la segunda parte: Generar varias instancias.

El código lo podemos consultar en colaborar/src/colaborar.java

Generamos el jar con botón derecho en el proyecto >clean and build

```
Created dir: /home/bobovino/Desktop/DAM/Segundo/Procesos/colaborar/dist
Copying 1 file to /home/bobovino/Desktop/DAM/Segundo/Procesos/colaborar/build
Nothing to copy.
Building jar: /home/bobovino/Desktop/DAM/Segundo/Procesos/colaborar/dist/colaborar.jar
To run this application from the command line without Ant, try:
java -jar "/home/bobovino/Desktop/DAM/Segundo/Procesos/colaborar/dist/colaborar.jar"
deploy:
jar:
BUILD SUCCESSFUL (total time: 0 seconds)
```

Y su respectivo javadoc

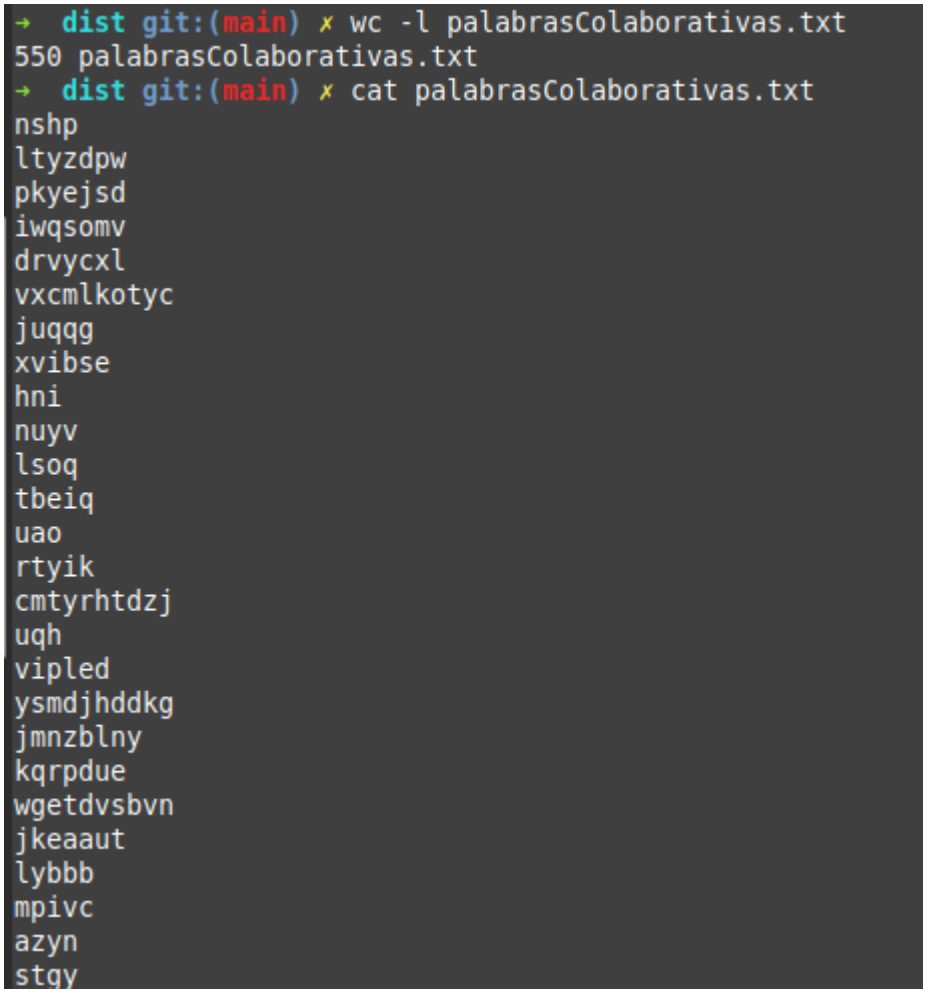
Klassen

Klasse	Beschreibung
colaborar	Lanza múltiples instancias de lenguaje para que trabajen juntas generando un fichero con palabras aleatorias.

Comprobamos en la terminal que el jar se ejecuta correctamente:

```
→  dist git:(main) ✗ ls
colaborar.jar  README.TXT
→  dist git:(main) ✗ java -jar colaborar.jar
Lanzando 10 procesos...

Proceso 1: 10 palabras
Proceso 2: 20 palabras
Proceso 3: 30 palabras
Proceso 4: 40 palabras
Proceso 5: 50 palabras
Proceso 6: 60 palabras
Proceso 7: 70 palabras
Proceso 8: 80 palabras
Proceso 9: 90 palabras
Proceso 10: 100 palabras

Esperando a que terminen...

Proceso 1 terminado (código 0)
Proceso 2 terminado (código 0)
Proceso 3 terminado (código 0)
Proceso 4 terminado (código 0)
Proceso 5 terminado (código 0)
Proceso 6 terminado (código 0)
Proceso 7 terminado (código 0)
Proceso 8 terminado (código 0)
Proceso 9 terminado (código 0)
Proceso 10 terminado (código 0)

=== FINALIZADO ===
Total esperado: 550 palabras
Fichero: palabrasColaborativas.txt

Verificar con: wc -l palabrasColaborativas.txt

Verificación: 550 palabrasColaborativas.txt
¡CORRECTO! 550 líneas
→  dist git:(main) ✗ wc -l palabrasColaborativas.txt
550 palabrasColaborativas.txt
```

Con wc contamos 550 palabras, que es el resultado esperado (si queremos otro resultado, cambiamos NUM_INSTANCIAS)



Y sigue hasta 550.

Con esto, también hemos comprobado que no hay interbloqueo, ya que todos los procesos terminan correctamente, ni inanición, ya que hay 550 palabras con datos íntegros ni líneas corruptas.