1 | Compiladores y intérpretes

Definición 1.1: Compilador

"Un compilador es un programa que lee un programa escrito en un lenguaje, el lenguaje *fuente*, y lo traduce a un programa equivalente en otro lenguaje, el lenguaje *objeto*. Como parte importante de este proceso de traducción, el compilador informa a su usuario de la presencia de [algunos] errores en el programa fuente."[a]

[a] Aho, Sethi y Ullman 1998, pp. 1.

Definición 1.2: Intérprete

Un intérprete traduce el lenguaje fuente al lenguaje objetivo y ejecuta el código en lenguaje objetivo conforme va realizando la traducción.

Meta

Que el alumno aprenda a utilizar un compilador para traducir código, detectar y corregir errores sintácticos y semánticos; y un intérprete para ejecutar `bytecode`.

Objetivos

Al finalizar la práctica el alumno será capaz de:

- Invocar al compilador de `Java`, `javac` desde una terminal para generar el *bytecode* ejecutable por la máquina virtual.
- Invocar a la máquina virtual de `Java` con el comando `java`, para ejecutar al código de una clase que contenga un método `main`.

- Empaquetar los archivos resultantes utilizando el comando `jar`.
- Ejecutar código en el archivo `.jar`.
- Generar la documentación del paquete utilizando `javadoc`.
- Utilizar la herramienta `ant` para compilar y empaquetar código, generar documentación y ejecutar un programa, utilizando un archivo `build.xml` provisto.

Desarrollo

Los programas de la JDK

Actividad 1.1

Descarga el archivo `practica1.tar.gz` y descomprímelo con el comando:

```
$ tar zxvf practica1.tar.gz
```

En el interior encontrarás dos carpetas: `Entrada` y `Reloj`. La primera parte de esta práctica se realizará con los archivos que se encuentran dentro de `Reloj`.

Actividad 1.2

Entra al directorio `src` y ejecuta los comandos siguientes:

```
1 $ javac icc/practica1/UsoReloj.java
```

Verás que aparecen una serie de errores, todos en el archivo `UsoReloj.java`. Abre ese archivo y utiliza los mensajes de error que te dio el compilador para corregir los errores. Puedes deducir con lógica qué es lo que debes hacer para lograrlo. Tendrás que invocar al compilador tantas veces como sea necesario hasta que ya no aparezcan errores.

Una vez que ya no haya errores, notarás que aparecieron archivos con terminación `.class` dentro de la carpeta `icc1/practica1`. Notarás que esos archivos tienen los mismos nombres que los archivos `.java` originales. Estos nuevos archivos contienen *bytecode*. Si intentas abrirlos con un editor de texto obtendrás una serie de símbolos ininteligibles, eso si no trabas a tu editor.

Escribe exactamente qué archivos fueron creados y dónde.

Actividad 1.3

Ahora invoca a la máquina virtual de `Java` para que interprete el código que generaste.

```
$ java icc.practica1.UsoReloj
```

Listo, ya tienes tu primer programa corriendo.

Actividad 1.4

Intenta invocar a la máquina virtual con los nombres de otros archivos `.class`. ¿Qué sucede? Lee lo que devuelve la consola y abre los archivos `.java` correspondientes que necesites. ¿Qué tiene el archivo `UsoReloj.java` que permite invocarl su `.class` con `java`?

Actividad 1.5

Ahora crearemos un archivo comprimido con solo el código ejecutable:

```
$ jar cvf UsoReloj.jar icc/practica1/*.class
```

Puedes ejecutar el programa almacendo en el `.jar` con:

```
$ java -cp UsoReloj.jar icc.practica1.UsoReloj
```

La opción `-cp` es una abreviatura de `classpath` y se usa para indicar a `Java` dónde buscar los archivos `.class`.

Para más información sobre el uso de `jar`, visita el `tutorial oficial`.

Actividad 1.6

Finalmente, ejecuta el comando siguiente:

```
$ javadoc icc.practica1
```

Esto generará una serie de archivos `.html` en el directorio donde te encuentras. Usa tu navegador de internet y abre el que se llama `index.html`. ¿Qué observas? Aquí están todos los comentarios en los archivos de código con los que estás trabajando, pero en un formato más amigable para el lector. Esta es la documentación que le

darás a tus usuarios cuando entregues tus trabajos.

Usando una herramienta auxiliar: ant

Aunque generaste todo lo necesario, el código compilado y la documentación se mezcló con los archivos con el código fuente. Al entregar un trabajo esto se ve desordenado y también te hará a tí más complicado el organizar tus archivos. A continuación lee la redacción de esta misma práctica del manual del Canek Peláez y Elisa Viso, para que repitas las mismas tareas, pero utilizando la herramienta `ant`. Las correcciones que realizaste al archivo `UsoReloj.java` son las mismas, así que guárdalas porque tendrás que entregar ese archivo. Cuando termines recuerda responder las preguntas que ahí se indican.

Ojo, tu archivo `build.xml` ha sido modificado ligeramente, por lo que los comandos con los que cuentas son:

compile	Compila la práctica.
run	Ejecuta la práctica, compilándola si no ha sido compilada.
doc	Genera la documentación JavaDoc de la práctica.
clean	Limpia la práctica de *bytecode*, documentación, o ambos.

Estructura de un programa

Para esta parte utilizarás el código dentro de la carpeta `Entrada`. El objetivo de esta sección es explicar cómo luce un programa sencillo en `Java` y descifrar algo del código que viste en los archivos de los ejercicios anteriores.

Juntos, el compilador `javac` y el intérprete `java` transforman el código que escribiste en secuencias de comandos que la computadora puede ejecutar como un programa.

Los programas de `Java` son muy parecidos a los programas que utilizas en la consola de Linux, como `ls`, `cd`, `pwd`, `more`, etc. En particular, también les puedes enviar parámetros al invocarlos, como harías al llamar `ls -al` o `diff archivo.txt archivo2.txt`.

Actividad 1.7

Entra a la carpeta `Entrada` e invoca `ant`. Te darás cuenta de que generó un archivo extra: `Entrada.jar`. Ignóralo por el momento y entra a la carpeta `build`. Usar los conocimientos adquiridos en las secciones anteriores para ejecutar desde aquí el programa `Entrada`. ¿Qué mensaje aparece?

Ahora ejecuta:

```
$ java icc.entrada.Entrada arg1 arg2 arg3
```

¿Qué obtienes?

Los archivos `.class` que se encuentran dentro de `build` fueron empaquetados en archivo `Entrada.jar`, por lo que es posible ejecutar el mismo programa utilizando sólo el `.jar`.

Actividad 1.8

Regresa al directorio que contiene el archivo `Entrada.jar` y ejecuta:

```
$ java -jar arg1 arg2 arg3
```

Prueba con diferentes argumentos y reporta lo que hace el programa.

Actividad 1.9

Ahora abre el código en `src/icc/entrada/Entrada.java` y lee cuidadosamente su documentación. Modifica el texto que se produce con cada argumento recibido. Entregarás tu archivo modificado como parte de esta práctica.

Actividad 1.10

Lee los dos archivos `build.xml` utilizados en esta práctica y observa en qué se parecen y en qué difieren. ¿Qué objetivos reconoce cada archivo? ¿Qué pasos ejecutará cada uno de los objetivos (observa el atributo llamado *depends*)?