1 | Compiladores e 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 es un programa que, una vez cargado en la memoria de una computadora y al ejecutarse, procede como sigue:

1. Toma un enunciado del programa en lenguaje de alto nivel, llamado código fuente.
2. Traduce ese enunciado y lo ejecuta.
3. Repite estas dos acciones hasta que alguna instrucción le indique que pare, o bien tenga un error fatal en la ejecución.

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

Si se te entrega un archivo comprimido: descarga el archivo `practica1.tar.gz` y descomprímelo con el comando:

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

Si tu código auxiliar se encuentra en un repositorio, clonalo en tu directorio de trabajo e ingresa a la carpeta correspondiente.

```
$ git clone <repo>
```

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:

```
$ 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 `icc/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.

Entregable: 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.

Entregable: Describe qué sucedió al intentar lo anterior. ¿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 almacenado 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:

```
1 $ 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 ti más complicado el organizar tus archivos. Además, al aumentar la complejidad de tu proyecto, la línea de comandos se vuelve más restrictiva y complicada de manejar. **Borra todos esos archivos extra antes de proseguir a la siguiente sección.** Si bien es posible arreglar este desorden aún sólo con los comandos de `Java` es mucho más sencillo utilizar una herramienta auxiliar: `ant`.

El programa `ant` utiliza un archivo de configuración, llamado `build.xml`, que le indica cómo realizar las acciones que le vamos a solicitar. Habrás notado la presencia de uno de estos archivos dentro de la carpeta `Reloj`, ahora lo vas a utilizar.

El archivo `build.xml` está escrito en *lenguaje de anotación extensible* (*eXtensible Markup Language XML*), este formato permite organizar información por medio de *etiquetas* de la forma `<etiqueta>` y sirve para muchos usos [1]. Su elemento distintivo es el uso de estas etiquetas anidadas para abrazar piezas de información: cada bloque inicia con una etiqueta de apertura `<etiqueta>` y al final se concluye con la de cierre `</etiqueta>`. En general, es posible poner más información o etiquetas dentro de cada par de etiquetas, donde la `etiqueta` que contiene a todas las demás se llama *etiqueta raíz*. Así mismo, una etiqueta también puede tener *atributos*. Un ejemplo genérico de esta gramática se muestra en el Listado 1.1.

[1] Valdés y Gurovich 2008

Listado 1.1: Forma genérica de un archivo XML

```xml
<?xml version="1.0" encoding="UTF-8"?>
<etiqueta atributo="valor" atributo2="valor2">
 <etq otroatributo="otrovalor">
 </etq>
 <etq>
 <!-- Comentario para explicar algo -->
 </etq>
 <!-- Abre y cierra etiqueta en versión abreviada -->
 <etq />
</etiqueta>
```

Para ant se ha definido un conjunto de etiquetas con el propósito de indicar procesos del JDK como *compilar*, *ejecutar* y *borrar archivos auxiliares*. El Listado 1.2 muestra un fragmento del archivo build.xml. La etiqueta que contiene a todas las demás, en este caso <project>, es la etiqueta *raíz* de este archivo. La etiqueta project tiene dos atributos que nos interesan particularmente: name que será el nombre de nuestro proyecto de Java y default que nos servirá para indicar la acción que realizará ant por defecto.

Listado 1.2: Fragmento de build.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<project name="practica1" default="compile" basedir=".">
 <target name="compile">
  ...
 </target>

 <target name="run" depends="compile">
  ...
 </target>

 ...
</project>
```

Las acciones que realizará ant se configuran mediante etiquetas <target>, donde se indica cómo invocar a los programas de la JDK. De esta manera, en el archivo build.xml se definen los comandos que tienes disponibles para trabajar con el proyecto que lo contiene. En este caso 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.

Actividad 1.7

Entra al directorio Reloj justo antes de src. Ahora compilarás tu código desde ahí usando ant. Observa que el archivo build.xml está justo a un lado. Escribe:

```
$ ant compile
```

Como ya corregiste los errores en `Reloj.java`, el proceso debe terminar si errores. Notarás que se crearon algunas cosas.

Entregable: ¿Dónde quedaron ahora los archivos `.class`?

Actividad 1.8

Ejecuta tu código con

```
$ ant run
```

¿Qué sucede?

Entregable: ¿A qué comando de los que usaste anteriormente correspondería?

Actividad 1.9

¿Recuerdas el desorden que provocó la documentación anteriormente? Prueba ahora con

```
$ ant doc
```

Actividad 1.10

El comando siguiente removerá todo lo que se generó al compilar con `ant` y será tu mejor amigo en el futuro.

```
$ ant clean
```

Después ejecuta de nuevo `ant compile` observando en la terminal el número de archivos que compila.

Una vez hecho esto ejecuta:[a]

```
$ touch src/icc/practica1/UsoReloj.java
```

Si quieres conocer más detalles sobre lo que hace `touch` escribe:

```
$ man touch
```

Ahora compila de nuevo con `ant compile` y observan cuátos y cuáles archivos se

compilan. ¿Qué notas diferente?

Entregable: ¿Qué mecanismo crees que utilice Ant para determinar qué compilar?

[a] El equivalente en Windows es `copy /b src\icc\practica1\UsoReloj.java +`

Como seguramente compilarás a menudo, pusimos a `compile` como la tarea por defecto. Entonces puedes compilar escribiendo simplemente:

```
$ ant
```

Pero si prefieres algún otro, sólo debes modificar el atributo `default` de la etiqueta `project`.

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 1.11

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`. Usa 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
```

Entregable: ¿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 1.12

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

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

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

Actividad 1.13

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.

Entregable: Tendrás que entregar el archivo con las modificaciones requeridas en esta actividad como parte de esta práctica.

Actividad 1.14

Lee los dos archivos `build.xml` utilizados en esta práctica y observa en qué se parecen y en qué difieren.

Entregable: ¿Qué objetivos reconoce cada archivo? ¿Qué pasos ejecutará cada uno de los objetivos (observa el atributo llamado *depends*)?

Entregables

Lo que deberás entregar para esta práctica es:

1. Archivo `UsoReloj.java` corregido.
2. Archivo `src/icc/entrada/Entrada.java` con la documentación completada.
3. Respuestas a las actividades:
   a) 1.2
   b) 1.4
   c) 1.7, 1.8 y 1.10
   d) 1.11 y 1.12
   e) 1.14
4. La respuesta, con su justificación, a la siguiente pregunta: Los errores que tuviste que corregir, ¿de qué tipo son, sintácticos o semánticos?