

# 1 | Compiladores e intérpretes

## Definición 1.1: Compilador

“Un compilador es un programa que lee un programa escrito en un lenguaje, el lenguaje *fuentes*, y lo traduce a un programa equivalente en otro lenguaje, el lenguaje *objetos*. Como parte importante de este proceso de traducción, el compilador informa a su usuario de la presencia de [algunos] errores en el programa fuente.”<sup>a</sup>

<sup>a</sup>Aho1998.

## 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 *byte-code* 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

Se te entrega el código necesario en un repositorio; clónalo 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:

```
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.

## 1. Compiladores e intérpretes

---

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.

```
1 $ 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 que sucedió al intentar lo anterior. ¿Qué tiene el archivo `Usoreloj.java` que permite invocarlo su `.class` con `java`?

---

### Actividad 1.5

Ahora crearemos un archivo comprimido con solo el código ejecutable:

```
1 $ jar cvf Usoreloj.jar icc/practica1/*.class
```

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

```
1 $ 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.practical
```

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. Además, al aumentar la complejidad de tu proyecto, la línea de comandos se vuelve más restrictiva y complicada de manejar.

En el archivo `build.xml` se definen los comandos que tienes disponibles para algún proyecto. En este caso tienes:

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

### Entregables:

1. Revierte algunas de las modificaciones que realizaste para que el proyecto compilara con errores y ejecuta el comando `ant compile`. Observa qué sucede y compáralo con lo que sucedió cuando usaste `javac`.
2. Ahora ejecuta el comando `ant run`. ¿Qué sucede? ¿A qué comando de los que usaste anteriormente correspondería?
3. Ejecuta `ant doc` y menciona qué diferencia observas con el resultado de generar la documentación con `javadoc`.
4. Ejecuta `ant clean`. Después ejecuta de nuevo `ant compile` observando en la terminal el número de archivos que compila.  
Una vez hecho esto ejecuta: `touch src/icc/practical1/Usoreloj.java` si te encuentras en MacOS o Linux; Si te encuentras en un sistema con Windows,

## 1. Compiladores e intérpretes

---

deberás ejecutar `copy /b src\icc\practical1\UsoReloj.java +,,`  
Ahora ejecuta `ant compile` de nuevo y observa cuántos archivos se compilan.  
¿Qué notas diferente? ¿Por qué crees que Ant se comporta de esta manera?

### 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`. Usa los conocimientos adquiridos en las secciones anteriores para ejecutar desde aquí el programa `Entrada`. ¿Qué mensaje aparece?

Ahora ejecuta:

```
1 $ 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 1.8

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

```
1 $ java -jar Entrada.jar arg1 arg2 arg3
```

**Entregable:** 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.

**Entregable:** Tendrás que entregar el archivo con las modificaciones requeridas en esta actividad 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.

**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 tus modificaciones.
3. Las respuestas a las preguntas descritas como entregables en las actividades:
  - a) 1.2
  - b) 1.4
  - c) 1.7 y 1.8
  - d) 1.10
4. Las preguntas a las preguntas entregables de la sección de ant.
5. La repuesta, con su justificación, a la siguiente pregunta: Los errores que tuviste que corregir, ¿de qué tipo son, sintácticos o semánticos?