Práctica: Ant y el compilador de Java

1

Every non-trivial program has at least one bug.

- *Corollary 1 - A sufficient condition for program triviality is that it have no bugs.*
- *Corollary 2 - At least one bug will be observed after the author leaves the organization.*

– Murphy's Laws of Computer Programming #1

Meta

Que el alumno comience a utilizar Ant para compilar, detectar errores de sintaxis y semánticos, y generar *bytecode* ejecutable.

Desarrollo

Java es un lenguaje *compilado*, lo que quiere decir que un compilador se encarga de transformar las instrucciones de alto nivel de un programa, en código que la Máquina Virtual de Java (Java Virtual Machine o JVM) puede ejecutar.

Actividad 1.1 Invoca al compilador de Java sin ningún argumento, con la siguiente línea de comandos:

```
# javac
```

(Nota que sólo debes escribir javac y después teclear Enter; el # sólo representa el *prompt* de tu intérprete de comandos).
Anota todas las opciones que se pueden pasar al compilador.

El compilador de Java no sólo genera el código ejecutable por la JVM; también detecta errores de sintaxis y semánticos, mostrando en qué línea ocurren.

Además, el compilador tiene lo que se conoce como *recuperación*; al encontrar el primer error no se detiene, trata de continuar tanto como sea posible y seguir encontrando errores. Esto último es importante porque le permite al programador corregir un mayor número de errores por cada compilación, en lugar de tener que compilar cada vez que quiere encontrar el siguiente error.

Ant

Es posible compilar cualquier proyecto de Java utilizando el compilador desde la línea de comandos. Sin embargo, conforme un proyecto crece en tamaño y complejidad, la línea de comandos va siendo cada vez más restrictiva.

Para ayudarnos con la compilación de proyectos en Java, utilizaremos *Ant*. Ant es un programa (escrito en Java, por cierto), que lee un archivo de configuración en XML, y de ahí obtiene la información necesaria para compilar un proyecto.

Actividad 1.2 Visita la página del proyecto Ant, en http://ant.apache.org/

Actividad 1.3 De la página de las prácticas[1], baja el archivo practica1.tar.gz, y descomprímelo con la siguiente línea de comandos:

```
# tar zxvf practica1.tar.gz
```

Después, trata de compilar la práctica:

```
# cd practica1
# ant compile
```

¿Cuántos errores marca? ¿Los entiendes? El significado de cada error está dado por la traducción del inglés.

El compilador de Java, a través de Ant, muestra en qué línea de un programa ocurre el error (si encuentra alguno). Si se utiliza un editor como XEmacs, y si está configurado de manera adecuada, se puede compilar dentro de XEmacs y saltar directamente a la línea en donde ocurre el error.

Actividad 1.4 Abre el archivo UsoReloj.java en XEmacs (está en el directorio icc1/practica1), y compila haciendo `C-c C-v C-b`, tecleando compile y dando `Enter` después.
Debe abrirse un segundo buffer en XEmacs donde se muestran los errores encontrados por el compilador. Cámbiate a ese buffer haciendo `C-x o` y teclea `Enter` en la primera línea que marque error. ¿Qué sucede?
En caso de que no funcione, puedes intentar cargar el archivo de proyecto que utiliza la práctica. Para esto, en XEmacs haz click en el menú JDE → Project → Project File → Load. Si no existe ningún menú JDE, tu XEmacs no está configurado como es debido.

Una vez que un programa está libre de errores, el compilador genera un archivo en *bytecode*.

La JVM ejecuta *bytecode*, un formato binario desarrollado para que los ejecutables de Java puedan ser utilizados en varias plataformas sin necesidad de recompilar.

El *bytecode* puede copiarse directamente a cualquier máquina que tenga una JVM, y ejecutarse sin cambios desde ahí. A eso se le conoce como *portabilidad a nivel binario*. Muchos otros lenguajes de programación tienen *portabilidad a nivel de código*, y otros no tienen ningún tipo de portabilidad.

[1] La página de las prácticas está disponible en http://abulafia.fciencias.unam.mx/practicas.

El archivo *build.xml*

Ant utiliza un archivo de configuración escrito en XML. XML es el *Lenguaje Extendible para el Formato de Documentos* (*Extensible Markup Language* en inglés), y es, sencillamente, una manera de representar información.

Actividad 1.5 Busca en la red información acerca de XML. Puedes comenzar en http://www.w3c.org

---

Para motivos de esta práctica, sólo es necesario saber que un documento XML tiene *etiquetas* (*tags*), que cada etiqueta tiene una *etiqueta de inicio* (del tipo <etiqueta>) y una *etiqueta final* (del tipo </etiqueta>), y que estas etiquetas están anidadas:

```
1 <etiqueta1>
2   <etiqueta2>
3    ...
4   </etiqueta2>
5   <etiqueta3>
6     <etiqueta2>...</etiqueta2>
7   </etiqueta3>
8    ...
9 </etiqueta1>
```

Esto es ilegal en XML:

```
<etiqueta1><etiqueta2>...</etiqueta1></etiqueta2>
```

No está anidado. Lo correcto es:

```
<etiqueta1><etiqueta2>...</etiqueta2></etiqueta1>
```

En XML, es equivalente escribir <et></et> a <et/>, para representar etiquetas "vacías". Todas las etiquetas de un documento XML pueden tener *atributos*, que son de la forma:

```
<etiqueta atributo1="valor1" atributo2="valor2">
```

La información que representa el archivo build.xml es la necesaria para manejar un proyecto en Java a través de Ant. Por ello, la etiqueta raíz del archivo build.xml (la etiqueta que envuelve a todas las demás), se llama *project* (*proyecto*).

En el archivo build.xml de esta práctica, dentro de la etiqueta raíz sólo hay etiquetas *target* (*objetivo*). Cada objetivo es una tarea que Ant puede ejecutar al ser llamado. Los objetivos de nuestro archivo build.xml son:

compile	Compila la práctica.
run	Ejecuta la práctica, compilándola si no ha sido compilada.
docs	Genera la documentación JavaDoc de la práctica (veremos esto la próxima práctica).
clean	Limpia la práctica de *bytecode*, documentación, o ambos.

Para decirle a Ant que ejecute un objetivo, sólo se lo pasamos como parámetro (tienen que hacer esto en el directorio donde esté el archivo build.xml):

```
# ant compile
# ant run
```

Si no se le pasa ningún objetivo a Ant, se ejecutará compile. Esto es porque la etiqueta project del archivo build.xml tiene un atributo llamado default, cuyo valor es compile:

```
2   <project name="practica1" default="compile" basedir=".">
```

La sintaxis del archivo build.xml la iremos analizando en las siguientes prácticas del curso.

Ejercicios

1. Corrige los errores que aparecen en la práctica, hasta que el programa compile y genere el *bytecode*. Utiliza los mensajes del compilador[2] para determinar la línea del error y en qué consiste. Todos los errores están en la clase UsoReloj, que está en el archivo UsoReloj.java (las clases de Java generalmente están en un archivo llamado como la clase, con extensión .java). *No* toques los demás archivos.

   (Puedes darle una mirada a los demás archivos de la práctica; sin embargo, el funcionamiento de estos archivos será discutido posteriormente.)

2. Una vez que el programa compile, ejecútalo con la siguiente línea de comandos (estando en el directorio practica1):

[2] Sí, los mensajes están en inglés. Ni modo.

```
# ant run
```

3. Ya que tu práctica compile y se ejecute como es debido, haz

   ```
   # ant clean
   ```

   para borrar los archivos que se hayan compilado. Después haz

   ```
   # ant
   ```

   y ve cuántos archivos dice ant que va a compilar (al momento de correr Ant con el objetivo compile, dice cuántos archivos se prepara a compilar).

   Una vez que termine de compilar, ejecuta el comando

   ```
   # touch src/icc1/practica1/UsoReloj.java
   ```

   y después vuelve a correr Ant. ¿Notas alguna diferencia? Explícala.

Preguntas

1. ¿Qué errores encontraste al compilar esta práctica? Explica en qué consisten.
2. Los errores que encontraste, ¿de qué tipo crees que sean, sintácticos o semánticos? Justifica tu respuesta.
3. ¿Cuántos archivos en *bytecode* (los que tienen extensión .class) se generaron?
4. ¿Cuál crees que sea la explicación del comportamiento de Ant después de hacer el ejercicio 3? Justifica tu respuesta.