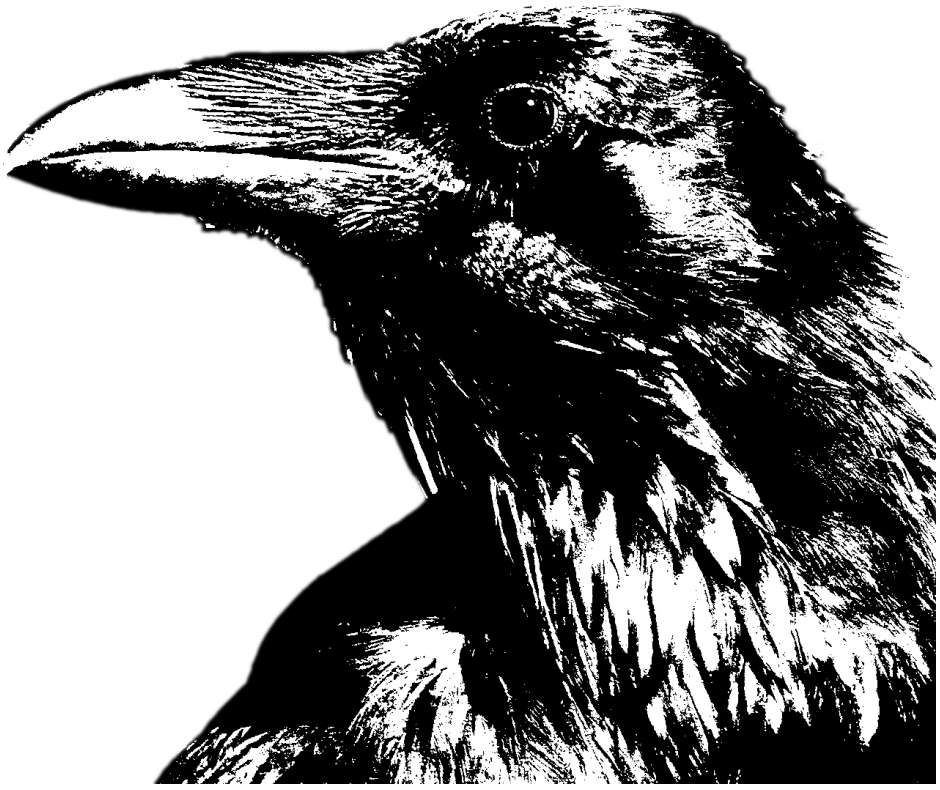
METODOLOGÍAS DE DISEÑO Y PROGRAMACIÓN ORIENTADA A OBJETOS

Basado en las clases de Alexandre Bergel

Ignacio Slater Muñoz



Departamento de Ciencias de la Computación
Universidad de Chile

Para mis alumnxs, de hoy y de siempre.

La parte del libro que nadie lee

La idea de este «apunte» nació como una *wiki* de *Github* creada por Juan-Pablo Silva como apoyo para el curso de *Metodologías de Diseño y Programación* dictado por el profesor Alexandre Bergel del Departamento de Ciencias de la Computación, Facultad de Ciencias Físicas y Matemáticas de la Universidad de Chile.

Lo que comenzó como unas notas para complementar las clases del profesor lentamente fue creciendo, motivado por esxs alumnxs que buscaban dónde encontrar soluciones para esas pequeñas dudas que no les dejaban avanzar.

El objetivo principal del texto sigue siendo el mismo, plantear explicaciones más detalladas, ejemplos alternativos a los vistos en clases y para dejar un documento al que lxs alumnxs puedan recurrir en cualquier momento.

Este libro no busca ser un reemplazo para las clases del curso, es y será siempre un complemento.

Esta obra va dirigida a los estudiantes de la facultad así como para cualquier persona que esté dando sus primeros pasos en programación. El libro presenta una introducción al diseño de software, la programación orientada a objetos y lo básico del lenguaje de programación *Java*. Se asume que los lectores tienen nociones básicas de programación, conocimiento básico de *Python* y, en menor medida, de *C*.

Antes de comenzar, debo agradecer a las personas que hicieron posible y motivaron la escritura de esto: Beatríz Graboloza, Dimitri Svandich y, por supuesto, Alexandre Bergel y Juan-Pablo Silva.

15 de marzo de 2021, Santiago, Chile

Índice general

Capítulo 1

Por algo se empieza

Este libro no partía así, y si alguien leyó una versión anterior se dará cuenta. Solía comenzar con una descripción e instrucciones para instalar las herramientas necesarias para seguir este libro y eso no estaba mal, pero no me agradaba comenzar así, simplemente presentando las herramientas sin ningún contexto de por qué ni para qué las ibamos a utilizar.

Puede parecer ridículo cambiar todo lo que ya había escrito solamente por eso, pero cuando nos enfrentamos a problemas del mundo real esto comienza a cobrar más sentido. Reescribo estos capítulos por una razón simple pero sumamente importante y que será una de las principales motivaciones para las decisiones de diseño que tomaremos a medida que avancemos, en el desarrollo de software **lo único constante es el *cambio***.[1] Una aplicación que no puede adaptarse a los cambios, sin importar que tan bien funcione, está destinada a morir.

¿Qué sucede entonces con las herramientas que vamos a utilizar? Las vamos a introducir, no podemos sacar una parte tan importante, pero no las vamos a presentar todas a la vez, en su lugar las iremos explicando a medida que las vayamos necesitando.

1.1. ¿Qué es un Java?

Java es uno de los lenguajes de programación más utilizados en el mundo (de ahí la necesidad de enseñarles éste y no otro lenguaje), se caracteriza por ser un lenguaje basado en clases, orientado a objetos, estática y fuertemente tipado, y (casi totalmente) independiente del sistema operativo.

¿Qué?

Tranquilos, vamos a ir de a poco. Comencemos por uno de los puntos que hizo que *Java* fuera adoptado tan ampliamente en la industria, la independencia del sistema operativo. Cuando *Sun Microsystems*[2] publicó la primera versión de *Java* (en 1996), los lenguajes de programación predo-

[1] Eric Freeman y Elizabeth Freeman, *Head First Design Patterns*.
[2] Actualmente *Java* es propiedad de *Oracle Corporation*

minantes eran *C* y *C++* (y en menor medida *Visual Basic* y *Perl*). Estos lenguajes tenían en común que interactuaban directamente con la API del sistema operativo, lo que implicaba que un programa escrito para un sistema *Windows* no funcionaría de la misma manera en un sistema *UNIX*. *Java* por su parte planteó una alternativa distinta, delegando la tarea de compilar y ejecutar los programas a una máquina virtual (más adelante veremos en más detalle parte del funcionamiento de la *JVM* para entender sus beneficios y desventajas). Esto último hizo que, en vez de cambiar el código del programa para crear una aplicación para uno u otro sistema operativo, lo que cambiaba era la versión de la *JVM* permitiendo así que un mismo código funcionara de la misma forma en cualquier plataforma capaz de correr la máquina virtual.[3] Pasarían varios años antes de que surgieran otros lenguajes que compartieran esa característica (destacando entre ellos *Python 2*, publicado el año 2000).

No tiene mucho sentido seguir hablando de *Java* si no podemos poner en práctica lo que vayamos aprendiendo, así que es un buen momento para instalarlo.

1.1.1. Instalando el *JDK*

El *Java Development Kit* (*JDK*) es un conjunto de herramientas que incluyen todo lo necesario para desarrollar aplicaciones en *Java* (la *JVM*, el compilador, la librería estándar, etc.), esto es lo que generalmente instalaremos si queremos programar en *Java*.[4]

Windows

Chocolatey Lo primero que necesitaremos para instalar las herramientas que usaremos será un gestor de paquetes, utilizaremos *Chocolatey*.[5]

Para partir abran una ventana de *Powershell* como administrador. Una vez abierta, deben ejecutar las instrucciones:

```
[Net.ServicePointManager]::SecurityProtocol = `
  [Net.SecurityProtocolType]::Tls12
Set-ExecutionPolicy -ExecutionPolicy RemoteSigned -Force
Invoke-WebRequest "https://chocolatey.org/install.ps1" `
  -UseBasicParsing | Invoke-Expression
```

Esto otorgará los permisos necesarios y descargará e instalará el gestor de paquetes.

[3] Actualmente casi todos los sistemas operativos son capaces de usar la *JVM*, en particular el sistema *Android* está implementado casi en su totalidad para usar esta máquina virtual.

[4] Existen otras herramientas que incluyen los contenidos de el *JDK* de forma total o parcial, por ejemplo: *Java SE*, *JRE*, o incluso otros lenguajes de programación que usan la *JVM*.

[5] *Chocolatey Software | Chocolatey - The package manager for Windows*.

(Opcional) Cmder Es sabido que las terminales por defecto de *Windows* dejan bastante que desear, por es una buena idea instalar una terminal externa (o más bien un emulador de una). Existen varias opciones, pero *Cmder* es una de las más completas.

1.1.2. Tipos en Java

Veamos ahora uno de los aspectos más importantes de la programación en *Java*, sobre todo viniendo de algún otro lenguaje de programación. *Java* es un lenguaje con tipos estáticos y fuertemente tipado, que tenga tipos estáticos significa que el tipo de todas las variables debe ser dado explícitamente al momento de definirla

1.1.3. Primeros pasos

Si programar fuera como salir a trotar (afortunadamente no lo es), podríamos pensar que todo hasta este punto fue la preparación previa, buscar las zapatillas, la ropa, poner su playlist más motivante. Pero todavía no podemos comenzar a hacer ejercicio, primero debemos calentar; eso es esta parte, un calentamiento para soltar las manos con *Java*.

Para los siguientes ejemplos puede serles útil tener dos terminales abiertas: una con *jshell* y otra con la consola interactiva de *Python*.[6]

Lo más básico que nos gustaría poder hacer en un programa es crear una función. Partamos por algo simple, imprimir un mensaje en pantalla, en *Python* esto se hace con la función `print(str)` mientras que en *Java* tenemos que usar `System.out.println(String)`, vemos inmediatamente que el segundo es bastante más complejo, por ahora ignoren la parte `System.out` y asuman que esa instrucción escribe en la salida estándar.[7]

```python
# Python
def jojo_reference(nombre):
    print("Mi nombre es " + nombre + " y tengo tuto")
```

```java
// Java
void jojoReference(String nombre) {
  System.out.println("Mi nombre es " + nombre + " y tengo tuto");
}
```

Revisemos los ejemplos en detalle. La primera línea en ambos ejemplos es un comentario, así como en *Python* los comentarios comienzan con `#`, en *Java* comienzan con `//`.

[6] Los comandos para abrirlos son `jshell` y `python` respectivamente.

[7] Si les ayuda, lo que hace la instrucción es escribir un mensaje en el canal de salida (*out*) del sistema, que no necesariamente va a ser la salida estándar siempre.

La siguiente línea es la *firma*[8] de la función. En *Python* las funciones se definen como:

```
def func_name(parameters):
```

sin necesidad de especificar los tipos por lo que explicamos en la sección anterior. En *Java* la sintaxis es bastante más explícita, teniendo que declarar el tipo de retorno de la función y el tipo de sus parámetros de la forma:

```
returnType funcName(Param1Type param1, Param2Type param2, ...) {...}
```

donde: (1) `returnType` es el tipo del valor que retorna la función, (2) `funcName` es el nombre de la función y, (3) `paramXType` y `paramX` son el tipo y nombre de cada parámetro. Otro detalle que podemos ver es que en *Python* la función se llama `func_name` y en *Java* `funcName`, esto es por que las convenciones de ambos lenguajes son distintas; pueden revisar las convenciones de *Java* en la sección **??**.

Por último está el cuerpo de la función. En *Python* el comienzo del cuerpo de una función se marca con `:` y el final de ésta está dada por la sangría (o *indentación*). En *Java* los espacios en blanco no tienen importancia para que el programa funcione,[9] en cambio, el inicio y fin de una función lo marcan la apertura y cierre de las llaves «`{}`». En general podremos pensar que en todos los casos en los que *Python* ocupa `:` el equivalente en *Java* serán las llaves (e.g. `if`, `while`).

[8] Esto lo veremos en detalle en la sección **??**.

[9] ¡Pero sí para hacer su código legible!

Bibliografía

Chocolatey Software | Chocolatey - The package manager for Windows **choco**

Chocolatey Software | Chocolatey - The package manager for Windows. URL: https://chocolatey.org (visitado 25-02-2021).

Evans y col.: Java in a Nutshell **java-nutshell-intro**

Benjamin J. Evans y David Flanagan. *Java in a Nutshell*. Ed. por Mike Loukides y Meghan Blanchette. 6th ed. O'Reilly Media, Inc, 2015. Cap. 1. Introduction to the Java Environment, págs. 3-15.

Eric Freeman y col.: Head First Design Patterns **head-first-intro**

Eric Freeman y Elizabeth Freeman. *Head First Design Patterns*. 1st ed. O'Reilly Media, Inc, 2004. Cap. Chapter 1: Intro to Design Patterns, págs. 1-35.