8 | Tipos de Datos y Clases

Verónica Esther Arriola Ríos

En esta práctica comenzarás haciendo uso de una clase ya definida en la *Interfaz de Programación de Aplicaciones*, API por sus siglas en inglés[1], de `Java`, para resolver un problema sencillo. En la segunda parte crearás tu propia clase, con atributos y métodos para resolver un problema un poco más complejo.

Meta

Que el alumno identifique las características que debe tener un tipo de dato abstracto, para que otros programadores hagan uso de él. Y que posteriormente comience a definir sus propios tipos de datos.

Objetivos

Al finalizar la práctica el alumno será capaz de:

- Utilizar objetos de clases definidas por otras personas.
- Definir sus propios tipos de datos para que los utilicen otros programadores.

Código Auxiliar 8.1: Cadenas

https://github.com/computacion-ciencias/icc-cadenas

[1]Application Programming Interface

Antecedentes

Diseño orientado a objetos

El diseño orientado a objetos es un paradigma para la organización de la información en un programa. Todo sistema programado con el paradigma orientado a objetos busca promover las siguientes cualidades Viso y Peláez 2007:

Modularidad Se deben trazar fronteras claras entre partes del sistema para que la tarea se pueda repartir entre varios programadores.

Bajo nivel de acoplamiento Cada módulo debe usar lo menos posible de otros módulos, de forma que puedan ser reutilizados en otras aplicaciones.

Alta cohesión Todos los elementos que se encuentren altamente relacionados entre sí, deben encontrarse dentro del mismo módulo. Esto hará que sean más fáciles de localizar, entender y modificar.

Para promover estas cualidades, un lenguaje orientado a objetos provee herramientas para definir *tipos de datos* que funjan como unidad administrativa de las piezas de información en un programa, junto con las operaciones que se pueden realizar sobre ellos.

Tipos de datos

Ya viste que un lenguaje de programación incluye un conjunto de tipos de datos básicos, a los cuales se les conoce como tipos de datos primitivos. Los lenguajes orientados a objetos, como `Java`, permiten que definas tus propios tipos de datos a partir de los que ya están definidos. El mecanismo para realizar este trabajo son las *clases* e *interfaces*.

El objetivo de las clases e interfaces es permitir la implementación de *tipos de datos abstractos*. La *interfaz pública*, es decir aquellos elementos que son visibles a los programadores que usan una clase o una interfaz, corresponde a la definición de un tipo de dato abstracto.

Definición 8.1: Tipo de dato abstracto

Un *tipo de dato abstracto* define un conjunto de valores y las operaciones que se pueden realizar sobre ellos, *sin especificar cómo serán implementados*.

De este modo, tanto la representación en la computadora de esos datos, como la implementación de las operaciones que sobre ellos se realizan, queda oculta a quien desea hacer uso de esos datos. Esto permite mantener la consistencia de los datos o modificar esas implementaciones sin alterar el código de quien los usa, entre otras cosas.

En `Java` las interfaces sólo te permiten definir las operaciones de un tipo abstracto de datos, estrictamente no puedes especificar el conjunto de datos. Las clases, por el contrario, te permiten declarar y definir tu nuevo tipo de datos. La correspondiencia es como sigue:

Conjunto de datos: está dado por los atributos de instancia y el conjunto de valores que se le pueden asignar a cada uno.

Operaciones: están definidas con la firma de los métodos. Son funciones cuyo primer argumento es siempre el objeto con el que se les manda llamar. Así mismo, puedes utilizar los comentarios de los métodos para establecer contratos con los usuarios de tu tipo de datos, es decir, aquí puedes especificar las precondiciones y poscondiciones que cumplirá tu implementación.

Cuando se aplica una operación sobre un objeto, se dice que se le envía un *mensaje*. El tipo de mensajes que se le pueden enviar a un objeto de un tipo dado, son las operaciones que se pueden hacer con él y están definidas por los métodos de esa clase.

Implementación: Al definir el contenido de los método, es decir, al programarlos, ya les estás dando una forma concreta en la computadora, lo que completa tu tipo de dato.

Cadenas

Para realizar este trabajo deberás utilizar una clase de `Java`: `String`. Esta clase representa secuencias de caracteres (tipo `char`). Cada caracter tiene un índice asociado dependiendo de la posición que ocupa en la cadena. Por ejemplo:

U	n	a	_	c	a	d	e	n	a
0	1	2	3	4	5	6	7	8	9

Dado que se trata de un tipo de datos especial, `Java` se tomó algunas libertades con ella. La forma más sencilla de crear variables tipo `String` es:

```
1 String cad = "Una cadena";
```

También puedes concatenar cadenas de dos formas: utilizando el operador + o con un método de la clase:

```
String nombre = "Pedro";
String saludo = "Hola␣" + nombre;
String saludo2 = "Hola␣".concat(nombre);
```

Las cadenas también tienen otra característica especial en Java: no es posible modificarlas. Una vez creadas ya no las puedes cambiar. Pero siempre es posible crear nuevas cadenas como segmentos de otras cadenas, con algún o algunos caracteres diferentes o uniendo cadenas.

Actividad 8.1

Revisa la documentación de la clase `String` en https://docs.oracle.com/javase/10/docs/api/java/lang/String.html. Pon especial atención a los métodos:

- `char charAt(int index)`
- `int indexOf(int ch, int fromIndex)` y similares
- `String replace(char oldChar, char newChar)`
- `String replaceAll(String regex, String replacement)`
- `String substring(int beginIndex, int endIndex)`

Caracteres a cadenas

Ojo, un `char` no es tipo `String`, pero puedes conseguir una cadena con un sólo caracter utilizando la concatenación de la cadena vacía ("") con un caracter:

```
String unChar = "" + 'a';
```

Este no es el método más elegante, pero sí el más práctico.

También puedes utilizar los métodos estáticos de la clase `String`:

```
String unChar = String.valueOf('a');
```

Observa que en este caso no tienes un objeto de tipo `String` con el cual mandes llamar a la función. Dada la funcionalidad de `valueOf` tiene sentido, pues quieres construir una cadena nueva a partir de un caracter y para ello no deberías necesitar a otra cadena. Este es un buen ejemplo del uso correcto de los métodos estáticos.

Entre los caracteres se utilizan códigos para representar símbolos especiales, como los caracteres `'\n'`, que significa *salto de línea* y `'\t'`, que representa un tabulador horizontal.

Cadenas iguales

Ahora que ya no estás trabajando con tipos primitivos debes tener cuidado. Java trabaja de forma diferente con los tipos primitivos y con las clases. Los valores primitivos pueden ser almacenados en cualquier región de la memoria, mientras que los objetos sólamente pueden residir en una: el montículo o *heap* en inglés.

Cuando declaras una variable de un tipo clase, tu variable no almacena al objeto, sólo tiene la dirección del objeto en el montículo. Así que si comparas dos variables con las direcciones de cadenas diferentes, aunque las cadenas sean iguales, Java te dirá que tienen valores diferentes.

```
String uno = "Hola";
String dos = "Hola";
boolean dif = (uno == dos)  // dif es false.
```

Para comparar a las cadenas, en vez de sus direcciones, debes utilizar el método `boolean equals(Object anObject)`.

```
dif = uno.equals(dos)       // dif es true.
```

Aprovechando, si quieres ordenar cadenas alfabéticamente puedes utilizar:

- `int compareTo(String anotherString)`.

Actividad 8.2

Revisa la documentación de los métodos de comparación:

- `boolean equals(Object anObject)`
- `boolean equalsIgnoreCase(String anotherString)`
- `int compareTo(String anotherString)`
- `int compareToIgnoreCase(String str)`

Entregable: ¿Qué resultaría de las siguientes comparaciónes? ¿Qué palabra va antes y cuál después?

```
"Hola".compareTo("Adios")
"Hola".compareTo("hola")
"Hola".compareToIgnoreCase("hola")
```

Cadenas a números

Para convertir cadenas a los números que representan tendrás que utilizar otros métodos:

- `static short parseShort(String s)`
- `static int parseInt(String s)`
- `static double parseDouble(String s)`

de las clases `Short`, `Integer` y `Double` respectivamente, y semejantes para los otros tipos primitivos.

Formatos

Hasta ahora, sólo hemos usado los métodos `println` y `print` para imprimir cadenas en la terminal. Sin embargo existe otra opción más interesante: `format`. Esta función nos permite utilizar una cadena como plantilla para el texto que se ha de imprimir, indicando las posiciones donde deseamos insertar información y pasar esta información en otros parámetros. Por ejemplo:

```
1 String producto = "manzanas";
2 float peso = 1.5f;
3 float precio = 45.86f;
4 System.out.format("Compré␣%.2f␣kg␣de␣%s␣a␣$%f%n", peso, producto,
    ↪ precio);
```

producirá la cadena:

```
Compré 1.50 kg de manzanas a $45.860001
```

El significado de los marcadores es el siguiente:

%.2f Número de punto flotante con dos dígitos de precisión.

%s Una cadena de texto.

%f Número de punto flotante.

%n Salto de línea. Funciona correctamente para cualquier sistema operativo.

Actividad 8.3

Consulta en la documentación de la clase `Formatter` las diferentes opciones que tienes para dar formato a las cadenas que imprimas.

Actividad 8.4

También puedes crear cadenas con formato, en lugar de imprimirlas, haciendo uso

del método `String.format`, revisa su documentación en la clase `String`. Notarás que se parece mucho a `Sytem.out.format`.

Programando clases

Las *clases* son plantillas que definen tipos de datos. Los *objetos* o *instancias* de la clase son ejemplares concretos. Así como el `5.6` es un ejemplar de `float`, la cadena `"Hola"` es un ejemplar de `String`.

Los tipos de datos que puedes definir están compuestos por otros tipos de datos definidos previamente. De este modo puedes definir el tipo `Círculo`, compuesto por las coordenadas de su centro y su radio.

Ejemplo 8.1. *Los atributos de instancia para cualquier círculo se definen dentro de la clase.*

Listado 8.1: Círculo.java dentro del paquete icc.clases

```
1 package icc.clases;
2
3 public class Círculo {
4   private double cx;
5   private double cy;
6   private double radio;
7 }
```

Estos datos solamente deben ser visibles para el círculo y nadie fuera de él, de modo que no puedan ser modificados desde fuera, por ello la declaración de estas variables va precedida por el acceso `private`. Así el círculo mantiene control sobre su estado y sólamente responde a través de *mensajes*. Los mensajes son las operaciones definidas para este tipo de datos y están determinados por los *métodos* de una clase; éstos son funciones que le pertenecen a los objetos de una clase y operan sobre sus atributos.

Un mensaje escencial para comenzar a trabajar con objetos es el método *constructor*, con el cual se asigna el valor inicial a los atributos del objeto. Dentro de los métodos nos podremos referir a este objeto con la palabra `this`. El método constructor es diferente pues no indica un tipo de regreso, éste está implícito: devuelve la dirección de un objeto recién creado, que se comportará según los lineamientos de la clase. El constructor se llama igual que la clase.

Otros dos tipos de métodos básicos para una clase son los *métodos de lectura* y *métodos de escritura*. Los métodos de lectura solicitan a la clase que nos permita leer el valor de uno de sus atributos. En inglés se les conoce como *getters* pues, si el atributo se llama `something`, su método de lectura se suele llamar `getSomething()`; como estamos programando en español usaremos una convención diferente: el método de lectura tendrá

el mismo nombre que el atributo pero llevará sus paréntesis que le corresponden como método. Así, para el atributo `algo`, su método de lectura se llamará `algo()`.

Los métodos de escritura (*setters* en inglés) nos permiten solicitarle a la clase que modifique el valor de uno de sus atributos. La ventaja de realizar esta operación a través de un método es que la clase puede asegurarse de que no cambiemos un valor sin que se entere, ni que asignemos valores inválidos. Estos métodos reciben como parámetro el nuevo valor que queremos asignar y no devuelven nada; si el valor es inválido lanzan una excepción. El ejemplo siguiente muestra uno.

Ejemplo 8.2. *Agregamos el constructor, un método para calcular el área del círculo y métodos de lectura y escritura para acceder a la información sobre el radio de forma segura.*

```
package icc.clases;

public class Círculo {
  private double cx;
  private double cy;
  private double radio;

  /** Inicializa un círculo. */
  public Círculo(double cx, double cy, double radio) {
    if (radio < 0) throw new
      IllegalArgumentException("No hay círculos con radio negativo")
        ↪ ;
    this.cx = cx;
    this.cy = cy;
    this.radio = radio;
  }

  /** Calcula el área. */
  public double calculaÁrea() {
    return Math.PI * this.radio * radio;
  }

  /** Método de lectura para el radio. */
  public double radio() {
    return radio;
  }

  /** Método de escritura para el radio.
    * Garantiza que no haya radios negativos. */
  public void radio(double radio) {
    if (radio < 0) throw new
      IllegalArgumentException("No hay círculos con radio negativo")
        ↪ ;
    this.radio = radio;
  }
}
```

Observa que el método `calculaÁrea` aparentemente no recibe parámetros. En realidad, todos los métodos de una clase reciben al menos un parámetro: la referencia al objeto sobre el cual debe actuar. Esta referencia es `this`. Para evitarnos escribirlo todo el tiempo, `Java` maneja el concepto de alcances: si dentro de un método se intenta usar una variable que no fue declarada en él, entonces asume que se trata de un atributo del objeto `this` y por lo tanto la busca entre las variables declaradas en la clase. Como ejemplo de esto, en el método `calculaÁrea` se invocó la variable `radio` tanto refiriéndose explícitamente al objeto `this`, como omitiéndolo. Ambas versiones funcionan. Ojo, dentro del constructor se muestra que, si hay variables locales que se llamen igual, entonces se vuelve obligatorio usar `this` para distinguirlas, pues en este caso se dice que las variables locales *ocultan* a los atributos.

Para crear ejemplares de una clase se utiliza `new`. Para enviarle un mensaje a un objeto, se utiliza el operador punto.

Ejemplo 8.3. *Creemos un objeto de tipo `Círculo` en el archivo de otra clase. Se requiere que esa clase esté en el mismo paquete que `Círculo`.*

Listado 8.2: UsoCírculo

```
1 package icc.clases;
2
3 public class UsoCírculo {
4   public static void main(String[] args) {
5     Círculo c = new Círculo(0, 0, 4);
6     double área = c.calculaÁrea();
7     System.out.format(El área del círculo es %.2f%n",␣área);
8 ␣␣}
9 }
```

Desarrollo

Una mosca parada en la pared

Tu primer ejercicio consistirá en hacer que unas moscas canten la canción "Una mosca parada en la pared". Para que no tengas que escribirla toda, ya está en el archivo `icc.clases.Mosca.java`, que se incluye con esta práctica. Como seguramente recordarás de tu infancia, el chiste de la canción radica en cambiar todas las vocales de la letra original por una sola vocal. Deberás programar el método `public String canta()`, donde la mosca deberá devolver la letra original modificada, según la vocal que se le haya indicado anteriomente con el método `public void setVocal(char vocal)`. Revisa cuidadosamente la documentación indicada anteriormente y lee los comentarios de la clase `Mosca`. Si tienes duda pregunta a tu ayudante. No debe tomarte más de seis líneas resolver este ejercicio utilizando correctamente los métodos de la clase `String` que se

mencionaron en las actividades.

RFC

Para la segunda parte de la práctica te toca implementar una clase llamada `Ciudadano`. Como atributos, cada ciudadano tiene:

- un `nombre`,
- un `apellidoPaterno` y
- un `apellidoMaterno` que se almacenan como tipo `String`,
- así como una fecha de nacimiento, que también es una cadena con el formato "dd/mm/aa"

El constructor debe recibir como parámetros estos datos y guardarlos en los atributos correspondientes. Recuerda, nadie debe cambiarle el nombre al ciudadano o su fecha de nacimiento, por lo que esos atributos deben ser `private`.

El RFC se calcula tomando la primera letra del apellido paterno más la primer vocal que le siga, la inicial del apellido materno y la inicial del nombre, seguido de los dígitos finales del año de nacimiento, los dos dígitos del mes y dos dígitos para el día. Para esta clase ignoraremos que los RFCs legales también agregan tres letras extra, conocidas como homoclave, que son asignadas por Hacienda para distinguir entre personas cuyos datos generen el mismo RFC.

Ejemplo 8.4. *Veamos aquí cómo se debe comportar un objeto de tipo `Ciudadano`, que puede calcular su RFC.*

Listado 8.3: UsoCiudadano.java

```
package icc.clases;

public class UsoCiudadano {
  public static void main(String[] args) {
    Ciudadano c = new Ciudadano("Laura Estrella",  // nombres
                                "León",            // apellido 1
                                "Calvo",           // apellido 2
                                "09/07/2000");     // fecha
    String rfc = c.calculaRFC();
    System.out.println("El RFC de Laura es " + rfc);
  }
}
```

```
$ java UsoCiudadano
El RFC de Laura es LECL000709
```

Ejercicios

Utilizando los métodos estudiados arriba, resuelve los ejercicios siguientes:

1. Implementa el método `public String canta();` de la clase `Mosca.java`. Al ejecutar `ant run_mosca` deberás ver el texto producido por dos moscas cantando la canción.
2. Implementa la clase `Ciudadano` con su constructor y su método `public String calculaRFC()`. Al ejecutar `ant run_rfc` deberás ver los nombres de tres mexicanos y sus RFCs calculados con tu programa.
3. Agrégale un metodo a tu ciudadano que se llame `String toString()` y devuelva una cadena con el nombre, cumpleaños y RFC del ciudadano. La clase `UsoCiudadano` lo utilizará para imprimir sus datos en la consola, descomenta el código de su método `main`[2].
4. Recuerda documentar el código de tu clase.

Entregables

Entrega la respuesta a la actividad 8.2 y completa los ejercicios de la sección anterior.

[2]Este código fue comentado para que puedas trabajar la parte de la canción sin tener que implementar `Ciudadano`, de otro modo el compilador te marcará errores.