# Capítulo 2. Getting to Know Java

. java → Compilamos → .class → Se unen en biblioteca → .jar

## Métodos:

* public → Se puede acceder mediante otras clases o métodos
* static → No necesita un objeto
* void → No devuelve ningún valor (set, test)
* (string[] args) → Para referirse a los argumentos del método.

## Naming Conventions:

* Clases → Primera letra en mayús. UpperCamelCase
* Variable de clases → lowerCamelCase
* Método → lowerCamelCase

## Tipo de Datos Primitivos:

* int → 0
* float → 0.0f
* double → 0.0d

## Operadores aritméticos:

* + // - // \* // / // %

## Operadores lógicos:

* && (and)
* || (or)
* ^ (True si uno de los dos es true)
* ! (not)

## Arrays

“tipo de dato (double, int, String)” [] “nombre variable”;

Para decir cuantos valor tendrá la array:

* “nombre variable” = new “tipo de dato” [5 (por ejemplo)]
* “nombre variable”[0] = “el valor que sea” (asignar valores a la array)

Y de forma más concisa:

* “tipo de dato”[] “nombre variable” = {1, 2, 3, …};

## Variable de clases

No cambian su valor (constante) a menos que lo cambiemos llamando a la variable y cambiemos nosotros

|  |
| --- |
| class Book{  static int maxPages = 500;  Cambiará cuando:  Book.maxPages = 2000; |

## Tipos de Variables

* Locales → Se declaran en un método
* De parámetro → Se declaran como argumentos de los métodos
* Instancia → Se declaran en una clase

## Métodos

### Tipos de métodos:

* De instancia → Pueden acceder los objetos
* De clase → Pertenecen a la clase como tal
* Constructores → Dice cómo se instancia una clase
* Principal → Para ejecutar la aplicación (main)

### Constructores

Se invoca cada vez que se crea una instancia, es decir, que se crea un nuevo objeto.

|  |
| --- |
| class Book{  final String title;  public Book(String t){  this.title = t;}  Book book1 = new Book(“Título libro”) |

## For

for (inicialización, condición, actualización) {statements}

## While

while (condition) {statements}

# 

# Capítulo 4. Hacia un programación orientada a objetos

double to Double → Boxing

Double to double → Unboxing

## Crear objetos

* new “nombre objeto” ();
* “tipo de objeto (Student)” “nombre variable (firstStudent)” = new Student();

## Variables

Instancia → Se utilizarán para contener datos de objetos

Variables de clases → Pertenece a la clase como tal, el palo del objeto.

Final → Variables/Constantes

### Instancia

Cada objeto mantiene una copia separada de esta variable y se puede variar

|  |
| --- |
| class Book{  String title;  String[] authors;}  Book book1 = new Book();  book1.title = “Java”;  book1.authors = new String[] {x1, x2, x3, ...)} |

Se puede poner valores predeterminados a las instancias

|  |
| --- |
| class Book{  String title = “Title1”;  String[] authors = new String[] {“Anonymous”} |

# Capítulo 7. Más conceptos de Programación orientada a objetos

## Sobrecarga de métodos

|  |
| --- |
| public class Book{  String title;  String author;  boolean isRead;  int numOfReadings;  public void read(){  this.isRead = true;  this.numOfReadings++;}  public void read (int i){  this.isRead = true;  this.numOfReadings += i;}} |

## Palabra clave THIS

|  |
| --- |
| public class Person{  String name;  public Person (String name){  this.name = name;}} |

## Encapsulación

* Público → Cualquier clase puede acceder
* Protected → Accesible mediante subclases o clases en el mismo paquete
* “Package” → Accesible a las clases del mismo paquete
* Private → Solo la misma clase puede acceder

Las clases solo pueden tener Publico o “Package”

## Getter

* Public
* Con tipo de dato
* Visualizar nombres de variables

## Setter

* Void
* Modificar valores de variables

|  |
| --- |
| public class Product{  private String productName;  private String productId;  private BigDecimal productPrice;  public Product(String productName, String id, BigDecimal price){  this.setProductName(name);  this.setProductId(id);  this.setPrice = price;}  public String getProductName(){  return this.productName;  public void setProductName(String name){  this.productName = name;}} |

## Herencia

|  |
| --- |
| public class Student extends Person{} |

### Super

* Forma de referirse a la superclase
* Se usa para invocar el constructor de la superclase
* Para acceder a las variables y métodos

|  |
| --- |
| public class Employee extends Person{  private Employee manager;  private int id;  public Employee (String name, Employee manager, int id){  super(name);  this.setManager(manager);  this.setId(id);}} |

## ToString

public String toString() → Representación de cadena del objeto

## Paquetes

* Herramienta organizativa para mantener juntas las clases seleccionadas
* El ClassPath le dice a Java donde encontrar el paquete y clases.

## Interfaz

* Public o “Package”
* Dice que debe hacer una clase pero no como hacerlo
* Puede tener variables solo si son finales (constantes)

# Capítulo 40. Arrays

Puede almacenar un solo tipo de valor (String, int…)

|  |
| --- |
| int[] numbers1 = new int[3]; → 3 espacios de memoria sin valor asignado  int[] numbers2 = {1, 2, 3} → Creamos un array con 3 espacios con valores asignados  int[] numbers3 = new int[] {1, 2, 3} → Inicializamos array |

* El tamaño de una array se fija cuando se inicializa.
  + Para poder modificar el tamaño debe usarse una clase collection.
* ArrayList almacena en una matriz y permite el cambio de tamaño.
* Cuando no se inicializa un array int, su valor predeterminado es 0 y si es una array string el valor predeterminado es null.

array.length → Devuelve el tamaño real de la matriz, no el número de valor que contiene

int[][] array = new int[5][4] → Crear array multidimensional

int[][] array = {{1,2}, {3, 4}, {5, 6}}

Arrays.fill(“array”, “valor predeterminado”) → Rellena un array con valor predeterminado

Arrays.fill(“array”, 1, 2, “valor predeterminado”) → Asignar valor a cada elemento del rango especificado. 1, 2 es el rango que tiene que tener el valor predeterminado

### Método setAll

Configurar cada elemento de una matriz a los valores generados

int[] array = new int[5]

Arrays.setAll(array”, i -> i)

array => {0, 1, 2, 3, 4}

### Declarando Arrays

int[] array9;

array9 = new int[3] → {0, 0, 0}

array9[0] = 10 → {10, 0, 0}]

array9[1] = 20 → {10, 20, 0}

### Crear lista a partir de Arrays

|  |
| --- |
| String[] stringArray = {“foo”, “bar”, “baz”}  List<String> stringList = Arrays.asList(stringArray) |

Cualquier cambio en la lista cambiará la matriz y viceversa, pero no se pueden cambiar de longitud.

ArrayList.addAll()

|  |
| --- |
| String[] stringArray = {“foo”, “bar”, “baz”}  List<String> list = new ArrayList<String>();  list.addAll(Arrays.asList(stringArray); |

Collections.addAll()

|  |
| --- |
| Collection.addAll(list, stringArray); |

El método Arrays.asList() no permite añadir o eliminar elementos.

stringList.add(“something”)

Se puede crear una lista pasando un array al constructor de una nueva lista.

List<String> modificableList = new ArrayList<>(Arrays.asList(“foo”, “bar”));

### Crear array a partir de coleccion

List<String> list = new ArrayList<>(1, 2, 3, 4, 5)

String[] stringArray = list.toArray();

### Método Stream.toArray()

String[] strings = list.stream().toArray(String[]:: new)

## Arrays a Stream

String[] arr = new String[]{“str1”,“str2”,“str3”}

Stream<String> stream = Arrays.stream(arr)

Los Array.stream() permite Arrays de valores primitivos y objetos.

### Iterando sobre Arrays

|  |
| --- |
| int[] array = new int10];  for (int i=0, i < array.length, i++){  array[i]= i;}  for (int e : array){  System.out.println(e);} |

### Iterando en un array multidimensional

|  |
| --- |
| int[][] array = new int[10][10];  for (int[] e : array){  for (int z : e){  System.out.println(z);}} |

### Arrays a Strings

|  |
| --- |
| Arrays.toString(Object[])  int[] arr = {1, 2, 3, 4, 5}  Arrays.toString(arr); |

### Ordenando Arrays

|  |
| --- |
| int[] array = {7, 4, 2, 1, 19}  Arrays.sort(array) → [1, 2, 4, 7, 19] |

### Longitud de una array

|  |
| --- |
| int[] arr1 = new int[10]  int len1 = arr1.length → 10 |

### Encontrar elemento en array

|  |
| --- |
| String[] strings = new String[] {“a”, “b”, “c”};  int[] ints = new int[] {1, 2, 3, 4};  int index = Arrays.binarySearch(strings, “a”);  int index = Arrays.asList(strings).indexOf(“a”);  int index = IntStream.range(0, strings.length).filter(i - > “a”.equals(Strings[i]). findFirst().orElse(-1)  boolean isPresent = Arrays.asList(strings)).contains(“a”). |

### Cambiar tamaño de un array

|  |
| --- |
| String[] listOfCities = new String[3];  listOfCities[0] = “NY”;  listOfCities[1] = “London”;  listOfCities[2] = “Berlin”;  String[] newArr = Arrays.copyOf(listOfCities, listOfCities.length + 1);  newArr[listOfCities.length] = “Sydney”; |

### Eliminar elementos de un array

* Se copia el array sin el/los elementos no deseados.
* Convertir de Array a Lista y después a Array con las modificaciones

|  |
| --- |
| String[] arr = new String[] {“foo”, “bar”, “baz”};  List<String> list = new ArrayList<>(Arrays.asList(arr));  list.remove(“foo”);  arr = list.toArray(new String[list.size()]); |

### Copiar arrays

|  |
| --- |
| int[] a = {4, 1, 3, 2};  int[] b = a.clone();  int[] b = Arrays.copyOf(a, a.length);  int[] b = new int[a.length];  System.arrayCopy(a, 0, b, 0, a.length); |

# Capítulo 13. Streams

* Un stream es una secuencia de elementos de datos.
* No tienen almacenamiento
* Se pueden ordenar o desordenar
* No se pueden reutilizar
* Un Stream extrae elementos de una fuente de datos y los pasa a una canalización para su procesamiento
* Los stream extraen los datos de colecciones o arrays
* Producen un resultado sin modificar la fuente de datos

|  |
| --- |
| ITERACCIÓN INTERNA  List <Integer> numbers = new Arrays.asList(1, 2, 3, 4, 5);  int sum = numbers.stream().filter(n -> n % 2 == 1).map(n -> n \* n).reduce(0, Integer::sum) |

### Stream para valores

|  |
| --- |
| String str = “Ken, Jeff, Chris, Ellen”;  Stream<String> stream = Stream.of(str.split(“,”)); |

* Los métodos accept() y add() agregan elementos al stream que se está construyendo
* Cuando se acaban de añadir elementos, se llama al método build() para crear el stream

|  |
| --- |
| Stream.Builder<String> builder = Stream.builder();  Stream<String> stream = builder.add(“Geeks”).build();  stream.forEach(System.out::println); |

### Interfaces IntStream

|  |
| --- |
| IntStream range (int start, int end) → No incluye el último  IntStream rangeClosed (int start, int end) → Incluye el último |

# Capítulo 57. Lists

* Una lista es una colección ordenada de valores

### Crear una lista

List<Integer> inte = new ArrayList<Integer>();

### Métodos de acceso posicional

* add(T type)
* add(int index, T type)
* remove(Object o)
* remove(int index)
* get(int index)
* set(int index, E element)
* int indexOf(Object o)
* int lastIndexOf(Object o)

### Ordenar una lista

|  |
| --- |
| sort(List<T> list, comparator <? super > c |

* Para ordenar una lista de objetos no estándar, tenemos que implementar la interfaz Comparabla <User> y sobreescrbirla para ordenar como deseemos.

|  |
| --- |
| public class User implements Comparable <User> {  public final Long id;  public final String username;  public User (Long id, String surname){  this.id = id;  this.username = username;}  @Override  public String toString(){  return String.format(“%s: %d”, username, id)  @Override  public int compareTo(User o){  return id.compareTo(o.id)}}  List<User> users = new ArrayList<User>();  users.add(new User(33L, “a”);  users.add(new User(25L, “b”);  users.add(new User(28L, “c”);  Collections.sort(users); → [b:25, c:28, a:33] |

### Convertir lista de enteros a lista de strings

|  |
| --- |
| List<Integer> nums = Arrays.asList(1, 2, 3);  List<String> strings = nums.stream().map(Object::toString).collect(Collectors.toList()); |

### ArrayList

Almacena los datos en una matriz y permite manipular el tamaño.

|  |
| --- |
| List<Integer> list = new ArrayList<Integer>(100);  int e1 = list.get(0) → 0  list.set(2, 10); → En la posición 2, set valor 10 |

List<T> myArrayList = new ArrayList<T>();

myArrayList.add(element) → Añadir elemento en la lista

myArrayList.add(index, element) → Añadir elemento en un index

myArrayList.remove(element) → Eliminar un elemento

myArrayList.remove(index) → Eliminar elemento a partir de su index LinkedList

|  |
| --- |
| List<Integer> list = new LinkedList<Integer>();  list.add(10);  list.remove() |

### Encontrar elementos comunes entre 2 listas

|  |
| --- |
| public static void main(String[] args){  List<Integer> numbersA = new ArrayList<Integer>();  List<Integer> numbersB = new ArrayList<Integer>();  numbersA.addAll(Arrays.asList(new Integer[]{1, 3, 4, 7, 5};  numbersB.addAll(Arrays.asList(new Integer[]{13, 32, 533, 3, 4};  List<Integer> numbersC = new ArrayList<>();  numbersC.addAll(numbersA);  numbersC.retainAll(numbersB); |

### Reemplazar Elementos

|  |
| --- |
| List<String> strings = new ArrayList<String>(“Inicio”, “Hola”, “Adios”);  strings.set(1, “Hi World”); |

|  |
| --- |
| int pos = strings.indexOf(“Adios”);  if (pos >= 0){  strings.set(pos, “Goodbye”); |

### Listas inmodificables

|  |
| --- |
| List<String> ls = new ArrayList<String>();  List<String> unmodifiableList = Collections.unmodifiablesList(ls); |

### Mover objetos de una lista

collections.revers(ls) → Revertir la lista

collections.rotate(ls, 1) → Mover los elementos 1 posición hacia delante

collections.shuffle(ls) → Mezclar elementos

Random()

Random random = new Random(12);

Collections.shuffle(ls, random);

### Iteraciones en listas

|  |
| --- |
| public void printEachElement(List<String> list){  for (String s: list){  System.out.println(s);}}  Para imprimirla todas en la misma línea usamos:  StringBuilder builder = new StringBuilder();  for (String s : list){  builder.append(s);}  System.out.print(builder.toString());} |

### Eliminar elementos de la lista b presentes en la lista A

List.removeAll(Collection c);

nombersB.removeAll(numbersA)