ESCUELA POLITÉCNICA SUPERIOR DE CÓRDOBA
Universidad de Córdoba

EPSC

UNIVERSIDAD D CÓRDOBA

Programación web

Prácticas

Semana 1: Introducción a Java y entorno de desarrollo

Aurora Ramírez Quesada (aramirez@uco.es)

Departamento de Ciencia de la Computación e Inteligencia Artificial

Universidad de Córdoba

Índice de contenido

Lenguaje de programación Java
Características del lenguaje

- **Orientación a objetos "pura"**. Cualquier función debe declararse dentro de una clase que heredará de java.lang.Object. No existen las estructuras (*struct*)
- **Simplicidad**. Elimina los punteros (todos los objetos por referencia), posee un recolector de basura (*Garbage Collector*), hay tipo *boolean*, etc.
- **Interpretado**. Los programas en Java (".java") se traducen a *bytecode* (".class") y la máquina virtual de Java (JVM) los interpreta pasándolos a código máquina propio de cada plataforma
- **Portable**. Se ejecuta en cualquier plataforma que disponga de una JVM (*Java Virtual Machine*) que interprete los archivos *bytecode* generados a código máquina

- Documentación oficial: https://docs.oracle.com/en/java/javase/24/docs/api/index.html
- Tutoriales para complementar (con ejemplos): https://www.w3schools.com/java

Lenguaje de programación Java
Tipos de datos

- Tipos primitivos
  - Numéricos: `byte`, `short`, `int`, `long`, `float`, `double`
  - No numéricos: `boolean`, `char`
- Todos los tipos tienen sus clases equivalentes (*Wrapper*):
  - Numéricos: `Byte`, `Short`, `Integer`, `Long`, `Float`, `Double`
  - No numéricos: `Boolean`, `Character`
- Casting entre tipos de datos primitivos:
  - Automático: Al convertir un dato a otro de mayor representación
  - Manual: Al convertir un dato a otro de menor representación
- Las clases (estáticas) proporcionan métodos para:
  - Creación: `Integer.valueOf(int i)`, `Integer.valueOf(String s)`
  - Transformación: `doubleValue()`, `floatValue()`, `parseInt()`
  - Heredados de `Object`: `equals(Object obj)`, `toString()`

```
int a = 5;
double b = a; // b=5.0
```

```
double b = 5.2;
int a = (int)b; // a=5
```

Lenguaje de programación Java
Tipos de datos

- Tipos no primitivos (referenciados)
  - **Clase**. Plantilla o abstracción que engloba un conjunto de instancias con propiedades similares
  - **Interfaz**. Entidad que define una colección de métodos que se deben implementar en una clase
  - **Array**. Colección de objetos del mismo tipo cuyo tamaño es fijo
  - **String**. Cadena de caracteres
- Java define una gran variedad de colecciones (listas, mapas, conjuntos) que permiten agrupar objetos de tipos compatibles
  - Las colecciones pueden trabajar con tipos básicos, pero solo si estos son definidos mediante clases (Wrapper)
  - https://www.w3schools.com/java/java_collections.asp

```
List<int> myList = new ArrayList<int>();
```

```
List<Integer> myList = new ArrayList<Integer>();
```

Lenguaje de programación Java
Tipos de datos: Arrays

- Se diferencia entre arrays como variable (que almacena lista fija de valores de un mismo tipo) y arrays como listas dinámicas:
  - Los arrays estáticos tienen la propiedad "*length*" y se accede a sus elementos con el operador []
  - Los arrays dinámicos se declaran como: `ArrayList<Type>`, donde `Type` debe ser una clase (no datos primitivos)
- Los `ArrayList` proporcionan métodos: `add`, `get`, `remove`, `set`, `size`

```
String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};
System.out.println(cars.length);
// Outputs 4
```

```
import java.util.ArrayList;

public class Main {
  public static void main(String[] args) {
    ArrayList<String> cars = new ArrayList<String>();
    cars.add("Volvo");
    cars.add("BMW");
    cars.add("Ford");
    cars.add("Mazda");
    System.out.println(cars);
  }
}
```

https://www.w3schools.com/java/java_arrays.asp
https://www.w3schools.com/java/java_arraylist.asp

Lenguaje de programación Java
Definición de clases

! Por convención, las clases se nombran con la primera letra en mayúscula

```
[<Modificadores>] class <NombreClase> [extends <ClasePadre>][implements <Interfaz>]
```

- [<Modificadores>]
  - **public**: la clase es accesible desde otras. Para acceder desde otros paquetes primero debe ser importada
  - **abstract**: tiene al menos un método abstracto y no puede ser instanciada
  - **final**: termina una cadena de herencia (no puede ser extendida)
- **extends**
  - Indica de qué clase hereda. Si no se especifica, por defecto heredará de **Object**
  - La herencia múltiple no es posible
- **implements**
  - Indica que la clase implementa una o varias interfaces
  - Las interfaces fuerzan a que un conjunto de métodos (cuya signatura es declarada en la interfaz) deban ser implementados en la clase (o en sus subclases, si es abstracta)

Lenguaje de programación Java
Definición de clases

- Cuerpo de la clase
  - Constructores. Si no se indica se crea uno por defecto
  - Variables. Accesibles dentro de toda la clase. Fuera depende de su visibilidad: `public`, `protected`, `private`. Generalmente privadas.
  - Métodos. Accesibles dentro de toda la clase. Fuera depende de su visibilidad: `public`, `protected`, `private`. Generalmente públicos.

```java
package es.uco.pw.demo;

public class Person {
    private String name;

    private int age;

    public Person(){
        this.name = "anonymous";
        this.age = -1;
    }

    public Person(String name, int age){
        this.name = name;
        this.age = age;
    }

    public String getName(){
        return this.name;
    }

    public int getAge(){
        return this.age;
    }

    @Override
    public String toString(){
        String message = "Hello! My name is " + this.name +
                         " and I'm " + this.age + " years old.";
        return message;
    }

}
```

- Ámbito
  - Clase (`static`) o instancia
  - Los métodos estáticos pueden simular las funciones de C/C++ no declaradas dentro de clases (para utilidades, por ejemplo, pero no conviene abusar)

Lenguaje de programación Java
Definición de clases

Modificador (métodos y variables)	Clase	Paquete	Subclase	Todos
public	Sí	Sí	Sí	Sí
protected	Sí	Sí	Sí	No
private	Sí	No	No	No
No especificado (a nivel de paquete)	Sí	Sí	No	No

Otros modificadores (avanzado): https://www.w3schools.com/java/java_modifiers.asp

Lenguaje de programación Java
Definición de interfaces

! Por convención, las interfaces se nombran comenzando con la letra "**I**"

```
[<Modificadores>] interface <NombreInterfaz> [extends <InterfazPadre>,…]
```

- [<Modificadores>]
  - **public**: las interfaces solo pueden ser públicas. Para acceder desde otros paquetes primero debe ser importada
  - **abstract**: aunque se pueda indicar, las interfaces son abstractas por definición
- **extends**
  - Indica que la interfaz hereda o deriva de otra(s)
  - La herencia múltiple sí es posible (separar con comas)

```java
// Interface
interface Animal {
  public void animalSound(); // interface method (does not have a body)
  public void sleep(); // interface method (does not have a body)
}

// Pig "implements" the Animal interface
class Pig implements Animal {
  public void animalSound() {
    // The body of animalSound() is provided here
    System.out.println("The pig says: wee wee");
  }
  public void sleep() {
    // The body of sleep() is provided here
    System.out.println("Zzz");
  }
}

class Main {
  public static void main(String[] args) {
    Pig myPig = new Pig();  // Create a Pig object
    myPig.animalSound();
    myPig.sleep();
  }
}
```

Ejemplo: https://www.w3schools.com/java/java_interface.asp

Lenguaje de programación Java
Definición de interfaces

- Cuerpo de la interfaz
  - Variables. Aunque no se especifique, son `public`, `final`, y `static`. Uso desaconsejado.
  - Métodos. Por defecto, son `public` y `abstract` (solo se indica su declaración)
  - Es posible implementar el cuerpo si se usan los modificadores `static` o `default`
- ¿Cuándo/cómo se usan?
  - Para separar abstracción (qué hacer) de implementación (cómo hacerlo)
  - Para mejorar la seguridad: solo "exhiben" comportamiento público, ocultan detalles
  - Puede ser una forma de conseguir la herencia múltiple a nivel de clases

```java
package es.uco.pw.demo;

public interface IMyInterface {
    public void method1();

    public default void method2(){
        System.out.println(x:"Default implementation");
    }
}
```

```java
package es.uco.pw.demo;

public class ClassFromInterface implements IMyInterface{

    @Override
    public void method1() {
        // TODO Auto-generated method stub
        throw new UnsupportedOperationException(message:"Unimplemented method 'method1'");
    }


    Run | Debug
    public static void main(String[] args) {
        ClassFromInterface myObject = new ClassFromInterface();
        myObject.method2();
        myObject.method1();
    }

}
```

```
Default implementation
Exception in thread "main" java.lang.UnsupportedOperationException: Unimplemented method 'method1'
        at es.uco.pw.demo.ClassFromInterface.method1(ClassFromInterface.java:8)
        at es.uco.pw.demo.ClassFromInterface.main(ClassFromInterface.java:15)
```

Lenguaje de programación Java
Estructuras de control

- Condicionales

¡Se desaconseja el uso de SWITCH!

```
if (<expresión-booleana>) {
   sentencias;
}
[else if (<expresión-booleana>) {
   sentencias;
}]
[else {
   sentencias;
}]
```

```
switch( <expresión> ) {
   case <valor1>:
        sentencias;
        break;
   case <valor2>:
        sentencias;
        break;
   [default:
        sentencias;]
}
```

Lenguaje de programación Java
Estructuras de control

- Bucles

```
while ( <expresión-booleana> ) {
    sentencias;
}
```

```
for ( <inicial>; <terminar>; <iteración> ) {
    sentencias;
}
```

```
do {
    sentencias;
} while ( <expresión-booleana> )
```

```
for (tipo nombreVar: nombreObjIterable ) {
    sentencias;
}
```

El bucle "forEach" permite iterar sobre los elementos de un objeto iterable (array, colecciones, etc.) sin tener que acceder a ellos explícitamente

```
String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};
for (String i : cars) {
  System.out.println(i);
}
```

Ejemplo: https://www.w3schools.com/java/java_foreach_loop.asp

Lenguaje de programación Java
Estructuras de control

- **Expresiones Lambda** + "**forEach**"
  - Incorporadas a la versión 8 de Java
  - Son pequeños fragmentos de código (similar a un método), pero que no necesitan tener un nombre ni ser implementados dentro de una clase
  - Se definen como: `parámetro(s) -> expresión`
  - Bloques simples de código (retorno, imprimir), no pueden contener variables, asignaciones o estructuras de control (`if`, `for`)
  - Para bloques más complejos, se debe usar: `parámetro(s) -> {código}`
  - Uso habitual: combinar con método "`forEach`" de `ArrayList` para aplicar el bloque de código a cada elemento del array iterativamente

```java
import java.util.ArrayList;

public class Main {
  public static void main(String[] args) {
    ArrayList<Integer> numbers = new ArrayList<Integer>();
    numbers.add(5);
    numbers.add(9);
    numbers.add(8);
    numbers.add(1);
    numbers.forEach( (n) -> { System.out.println(n); } );
  }
}
```

Ejemplo: https://www.w3schools.com/java/java_lambda.asp

Lenguaje de programación Java
Excepciones

- Una excepción es una condición anormal que surge en una secuencia de código durante su ejecución
- Mecanismo para el tratamiento de errores, que permite a los métodos no finalizar abruptamente ante una situación anómala

```
try {
     // Código que puede generar una excepción;
}
catch( TipoExcepción e ) {
     // Código para tratar la excepción;
}

[catch(...) ]

[finally {
     // Código para finalizar el bloque;
}]
```

Lenguaje de programación Java
Excepciones

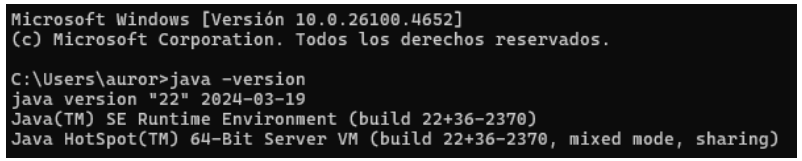
- Pueden definirse nuevas excepciones a partir de las existentes
- Si no son capturadas cuando ocurren, se propagan hasta el programa principal (pila)
- Pueden lanzarse explícitamente mediante `throw new Exception`
- Errores y excepciones comunes en Java

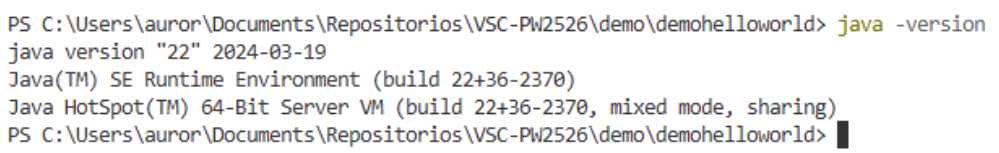
Error/Exception	Description
ArithmeticError	Occurs when a numeric calculation goes wrong
ArrayIndexOutOfBoundsException	Occurs when trying to access an index number that does not exist in an array
ClassFormatError	Occurs when a class file cannot be accessed
ClassNotFoundException	Occurs when trying to access a class that does not exist
ConcurrentModificationException	Occurs when an element is added or removed from iterables
FileNotFoundException	Occurs when a file cannot be accessed
IncompatibleClassChangeError	Occurs when there's been a change in a base class after a child class has already been initialized
InputMismatchException	Occurs when entering wrong input (e.g. text in a numerical input)
InterruptedException	Occurs when a Thread is interrupted while waiting/sleeping
InvalidClassException	Occurs when the Serialization runtime observes a problem with a class
IOException	Occurs when an input or output operation fails
NegativeArraySizeException	Occurs when trying to create an array with negative size
NoClassDefFoundError	Occurs when the class is not found at runtime
NoSuchFieldException	Occurs when trying to access a class field/variable that does not exist
NoSuchMethodException	Occurs when trying to access a class method that does not exist
NullPointerException	Occurs when trying to access an object referece that is `null`
NumberFormatException	Occurs when it is not possible to convert a specified string to a numeric type
RuntimeException	Occurs when an exception occurs at runtime
StringIndexOutOfBoundsException	Occurs when trying to access a character in a String that does not exist
TypeNotPresentException	Occurs when a type cannot be found
IllegalArgumentException	Occurs when when an illegal argument is passed to a method
IllegalStateException	Occurs when when a method is called at an illegal time

https://www.w3schools.com/java/java_ref_errors.asp

Entorno de desarrollo
Instalación de JDK/JVM

- El entorno Java se compone de tres elementos:
  - *Java Virtual Machine* (JVM): convierte el *bytecode* en el código máquina específico
  - *Java Runtime Environment* (JRE): se encarga de la ejecución de programas Java
  - *Java Development Kit* (JDK): permite el desarrollo de aplicaciones Java
- Tanto JRE como JDK incluyen JVM
- Última versión de Oracle es la v24; Existe versión en abierto: OpenJDK
- Algunos IDE pueden traer una versión instalada, o facilitar su instalación a través del "*marketplace*". Si se instala "*standalone*", la reconocen automáticamente
- Para verificar la instalación de Java (y ver la versión): `java -version`
- Si no se configura automáticamente, es necesario indicar la ruta con la variable **`Path`** o **`JAVA_HOME`** (según sistema operativo)





Entorno de desarrollo
Visual Studio Code para Java

- Se recomienda el uso de **Visual Studio Code** para el desarrollo de las prácticas
- Soporte para la programación en Java mediante extensiones específicas
- Facilita la edición, ejecución y depuración de aplicaciones Java
- Fácil integración de componentes para desarrollo web: Maven, Tomcat, Spring Boot
- Control de versiones con Git
- Se recomienda instalar el "*Coding Pack for Java*" (Windows, Mac) o "*Extension Pack for Java*" (otros SO)
- Si se tienen varias versiones de JDK, se puede configurar cuál utilizar: https://code.visualstudio.com/docs/java/java-project#_configure-runtime-for-projects
- Más información: https://code.visualstudio.com/docs/languages/java#_install-visual-studio-code-for-java

Entorno de desarrollo
Instalación y configuración de Spring

- Framework Java afianzado para desarrollo web: https://spring.io/
- Facilita el desarrollo, automatizando ciertos componentes (configuración, acceso a datos, etc.)
- Permite el desarrollo de diferentes arquitecturas (MVC, microservicios, API REST, etc.)
- Documentación extensa, tutoriales y guías: https://spring.io/guides
- Libro de referencia (disponible en biblioteca):

C. Walls, Spring in Action (6th edition).

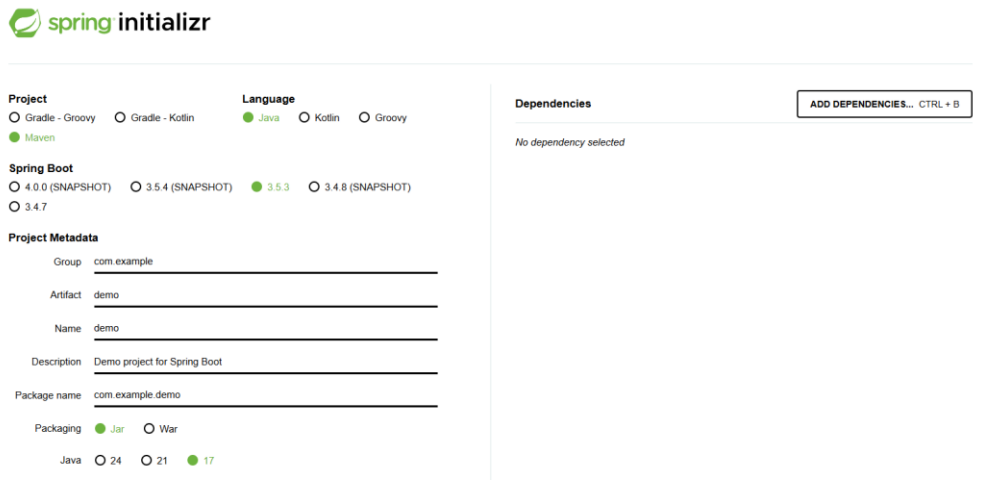
Manning. 2022

Entorno de desarrollo
Instalación y configuración de Spring

- VSC proporciona un conjunto de extensiones para el desarrollo de aplicaciones con Spring Boot:
  - Spring Boot Tools
  - Spring Initializr Java
  - Spring Boot Dashboard

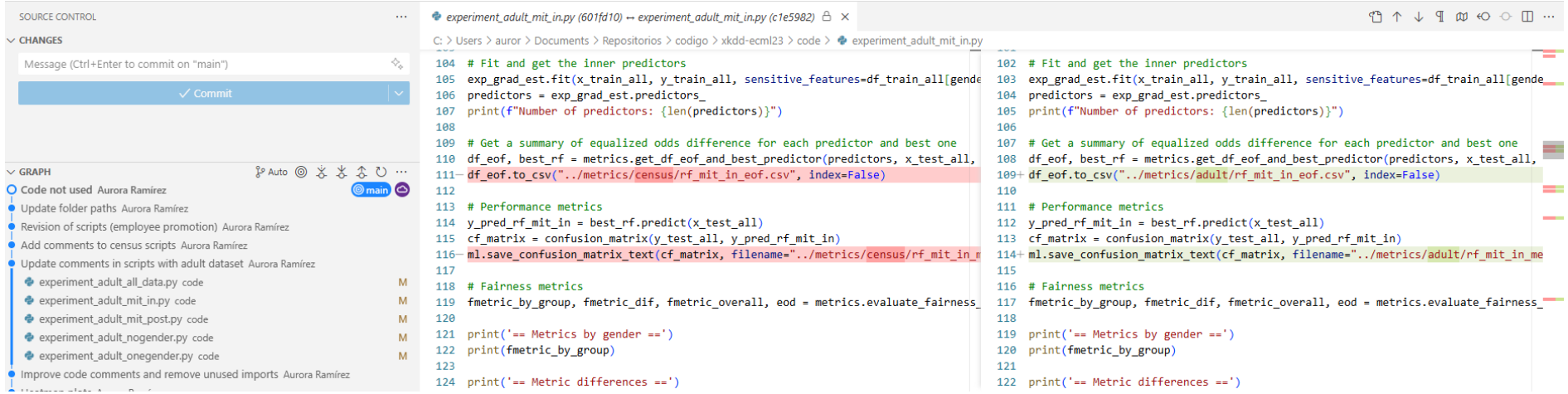
  https://marketplace.visualstudio.com/items?itemName=vmware.vscode-boot-dev-pack

- Plataforma para configuración de proyectos Spring Boot: https://start.spring.io/
  - Crea un proyecto (ZIP) con la estructura de directorios necesaria
  - Incorpora las dependencias (Maven) que se necesiten según el tipo de proyecto



Entorno de desarrollo
Control de versiones con Git/GitHub

- Indispensable para desarrollo colaborativo de software
- Facilita la trazabilidad del código y recuperación de versiones
- Integrado en VSC: Historial de *commits*, subir cambios, ver versiones…
- Más información:
  - https://code.visualstudio.com/docs/sourcecontrol/overview
  - https://code.visualstudio.com/docs/sourcecontrol/intro-to-git



Entorno de desarrollo
Control de versiones con Git/GitHub

- Versión remota alojada en GitHub
  - Herramientas adicionales para gestión de *issues*, wiki, etc.
  - Entorno más “amigable” para visualizar el histórico de desarrollo, contribuidores, cambios en ficheros, etc.
- Más información:
  - Cómo usar Git/Github en VSC:

    https://code.visualstudio.com/docs/sourcecontrol/github
  - Cómo invitar a colaboradores:

    https://docs.github.com/es/account-and-profile/setting-up-and-managing-your-personal-account-on-github/managing-access-to-your-personal-repositories/inviting-collaborators-to-a-personal-repository

Invitar a un colaborador a un repositorio personal

Puede enviar una invitación para colaborar directamente en el repositorio a alguien en GitHub, o a la dirección de correo electrónico de la persona.

GitHub limita la cantidad de personas a las que se puede invitar a un repositorio en un período de 24 horas. Si excedes este límite, espera 24 horas o crea una organización para colaborar con más personas. Para más información, consulta Crear una organización nueva desde cero.

1. Solicite el nombre de usuario de la persona que está invitando como colaboradora. Si todavía no cuentan con nombre de usuario, pueden registrarse en GitHub. Para más información, consulta Creación de una cuenta en GitHub. 1. En GitHub, navegue hasta la página principal del repositorio.
2. En el nombre del repositorio, haz clic en **Configuración**. Si no puedes ver la pestaña "Configuración", selecciona el menú desplegable ··· y, a continuación, haz clic en **Configuración**.

3. En la sección "Acceso" de la barra lateral, haz clic en **Colaboradores**.
4. Haz clic en **Agregar personas**.
5. Comienza a teclear el nombre de la persona que deseas invitar dentro del campo de búsqueda. Posteriormente, da clic en algún nombre de la lista de coincidencias.
6. Haga clic en **Agregar NOMBRE al REPOSITORIO**.
7. El usuario recibirá un correo electrónico invitándolo al repositorio. Una vez que acepte la invitación, tendrá acceso de colaborador a tu repositorio.

El repositorio colaborativo será el **eje central** del trabajo en equipo. Es indispensable crearlo, mantenerlo actualizado y utilizarlo profesionalmente

Objetivos de la semana

1. Instalación de JDK (salvo máquinas UCO)
   - Por compatibilidad con las máquinas UCO, se recomienda instalar la v17
2. Instalación de Visual Studio Code (salvo máquinas UCO): https://code.visualstudio.com/
3. Instalación de extensiones para VSC (salvo máquinas UCO):
   - Coding Pack for Java
   - Spring Boot Extension Pack
4. Crear proyecto “Hello World” en Spring siguiendo la guía: https://spring.io/quickstart
   - Importante: Seleccionar Maven y versión Java compatible con la instalada
5. Instalación y configuración de Git (salvo máquinas UCO): https://git-scm.com/
6. Creación de un repositorio privado compartido por equipo de prácticas
   - El fichero **Readme.md** debe identificar a los integrantes del equipo y el GM al que pertenecen
   - Incluir a la profesora como colaboradora **(aramirez-uco)**
7. Practicar Java (p. ej.): https://www.w3schools.com/java/java_exercises.asp