Actividad 6.2: Introspección y Reflexión

A. Investigar sobre la característica de "Reflexión" que ofrece JAVA

- **Brevemente, ¿qué es y para qué sirve la característica de JAVA llamada Introspección? ¿y la característica de JAVA llamada Reflexión? ¿Qué paquete java ofrece esas características?**

  La reflexión y la introspección permiten hacer los mismo sin conocer la clase al compilar: el programa "se mira si mismo" mientras se ejecuta.

  - **Reflexión:** es la capacidad de un programa para examinar y modificar en tiempo de ejecución las clases cargadas en la máquina virtual: ver sus atributos, métodos y constructores, crear objetos y llamar a métodos. Es una característica avanzada que permite examinar o modificar el comportamiento de las aplicaciones que se ejecutan en la JVM.

  - **Introspección:** es una forma de reflexión especializada en JavaBeans. En lugar de ver métodos sueltos, entiende propiedades: si ve getCodigo() y setCodigo(), deduce que existe la propiedad código. La clase Instrospector ofrece una forma estándar de conocer las propiedades, eventos y métodos de un vean, y por dentro usa reflexión para analizar la clase.

  Paquetes:

Característica	Paquete	Clases principales
Reflexión	`java.lang.reflect` (+ `java.lang.Class`)	`Class`, `Field`, `Method`, `Constructor`, `Modifier`
Introspección	`java.beans`	`Introspector`, `BeanInfo`, `PropertyDescriptor`, `MethodDescriptor`

- **¿Cuáles son los principales métodos que se usan en Introspección?**

Método	Qué hace
`Introspector.getBeanInfo(Clase.class)`	Analiza el bean y devuelve un `BeanInfo`
`Introspector.getBeanInfo(Clase.class, Object.class)`	Igual, pero sin incluir lo heredado de `Object` (evita la propiedad `class`)
`beanInfo.getPropertyDescriptors()`	Propiedades del bean
`beanInfo.getMethodDescriptors()`	Métodos públicos del bean
`pd.getName()` / `pd.getPropertyType()`	Nombre y tipo de la propiedad
`pd.getReadMethod()` / `pd.getWriteMethod()`	Su getter y su setter (como objetos `Method`)

- **¿Cuáles son los principales métodos que se usan en Reflexión?**

Método	Qué hace
`Class.forName("paquete.Clase")`	Carga una clase a partir de su nombre en texto
`objeto.getClass()` / `Clase.class`	Obtiene el objeto `Class`
`getName()` / `getSimpleName()` / `getPackageName()`	Nombre completo, nombre corto y paquete
`getDeclaredFields()` / `getFields()`	Atributos (todos los declarados / solo los públicos, incluidos heredados)
`getDeclaredMethods()` / `getMethods()`	Métodos (mismo criterio)
`getDeclaredConstructors()` / `getDeclaredConstructor(tipos...)`	Constructores
`getMethod("nombre", tipos...)`	Un método público concreto
`constructor.newInstance(args...)`	Crea un objeto
`metodo.invoke(objeto, args...)`	Ejecuta un método sobre un objeto
`campo.get(obj)` / `campo.set(obj, valor)`	Lee o modifica un atributo
`setAccessible(true)`	Permite acceder a miembros `private`
`Modifier.toString(miembro.getModifiers())`	Muestra los modificadores (`public`, `private`, `static`...)

- **Busca un ejemplo sencillo de uso.**
  // Reflexión: ver los atributos de una clase
  Class clase = Continente.class; for (Field campo : clase.getDeclaredFields()) { System.out.println(campo.getName() + " (" + campo.getType().getSimpleName() + ")" }

  // Introspección: ver las propiedades del vean

  BeanInfo info = Introspector.getBeanInfo(Continente.class, Object.class); for (PropertyDescriptor pd : info.getPropertyDescriptors()) { System.out.println(pd.getName() + " -> " + pd.getPropertyType().getSimpleName()) }

- **¿Qué harías si tuvieras el nombre de una clase como "literal" (con su paquete incluido) y necesitas crear un objeto de esa clase en tiempo de ejecución?**

  String nombreClase = "org.palomafp.aadd.npz.model.Continente";
  Class clase = Class.forName(nombreClase);
  Object obj = clase.getDeclaredConstructor().newInstance();

  - Class.forName() lanza ClassNotFoundException si la clase no existe.
  - getDeclaredConstructor() sin parámetros busca el constructor vacío. Por eso los JavaBeans y las entidades deben tenerlo.
  - La forma antigua clase.newInstance() está obsoleta desde Java 9; se usa la de arriba.
  - Ejemplo real en Acceso a Datos: Class.forName("com.mysql.cj.jdbc.Driver"); para cargar el driver JDBC.

- **¿Puedes invocar un método de un objeto sin conocerlo en tiempo de compilación? ¿Cómo?**

  Si, con getMethod() + invoke():

```
Method setter = clase.getMethod("setCodigo", String.class);
setter.invoke(obj, "01");
Method getter = clase.getMethod("getCodigo");
Object codigo = getter.invoke(obj);
```

Hay que indicar los tipos de los parámetros porque puede haber sobrecarga. invoke devuelve siempre Object (y null si el método es void). Como el nombre del método es un String, podría venir de un fichero, una base de datos o lo que escriba el usuario. Todas las excepciones (ClassNotFoundException, NoSuchMethodException, IllegalAccessException, InvocationTargetException...) heredan de ReflectiveOperationException, así que se pueden capturar juntas.

- **¿Qué riesgos o desventajas crees que tiene usar reflexión en exceso?**
  1. Errores en ejecución, no en compilación: si escribo "setCodgo", Eclipse no avisa; falla al ejecutar con NoSuchMethodException.
  2. Peor rendimiento: buscar e invocar métodos por nombre es más lento que una llamada normal.
  3. Rompe la encapsulación: con setAccessible(true) se modifican atributos private. Oracle advierte de que acceder así a miembros privados puede causar efectos inesperados, dejar el código inservible y romper la portabilidad (Oracle, The Reflection API).
  4. Código frágil: si alguien renombra un método, el código reflexivo deja de funcionar sin que nadie lo note; además, puede cambiar de comportamiento al actualizar la plataforma (Oracle, The Reflection API).
  5. Difícil de leer y depurar: m.invoke(obj, x) dice mucho menos que continente.setCodigo(x).

- Dentro de acceso a datos (2º DAM), indícame 3 ejemplos de proyectos populares en los que se esté utilizando la introspección y la reflexión.

    1. **Hibernate / JPA (ORM):** al leer una fila de T_CONTINENTE, Hibernate crea el objeto Continente con su constructor vacío por reflexión y rellena los atributos con las columnas. También lee por reflexión las anotaciones @Entity, @Id o @Column. Por eso la especificación exige que toda entidad tenga un constructor sin argumentos public o protected.
    2. **Jackson / JAXB (JSON y XML):** inspeccionan la clase, descubren sus propiedades a través de los getters y generan el fichero; al leer, crean los objetos y llaman a los setters. Es introspección pura, muy ligada al tema de ficheros XML de la asignatura.
    3. **Spring / Spring Data JPA:** Spring crea los objetos (por ejemplo, los DAO) y los conecta entre sí por reflexión. Spring Data lee por reflexión el nombre de métodos como findByNombre(String nombre) y genera la consulta automáticamente.

B. Aplicación práctica de Introspección y Reflexión

Crea un proyecto en el que tenemos una clase de persistencia Continente. Se pide crear una clase principal que permita:

- Utilizando introspección imprimir los métodos y atributos de la
- clase Continente.
- A partir de un literal con el nombre de la clase Continente
- (org.palomafp.aadd.xxx.model.Continente) se quiere:
    - Construir un objeto de tipo Continente sin usar directamente cualquiera de los constructores que tiene (por Reflexión)
    - Cargar el objeto asignando un valor al código.
    - Imprimir el objeto previamente cargado.