ESCUELA DE SISTEMAS Y TECNOLOGÍAS

Transparencias de **ANALISTA DE SISTEMAS**

Edición 2020 Materia: **Java Web**

TEMA: Reflection

Agenda

- Introducción a Reflection
- API de Reflection

Introducción a Reflection (1)

- Reflection es un API incluida en Java (en el package **java.lang.reflect**) cuya utilidad es añadir al lenguaje la posibilidad de utilizar el paradigma de metaprogramación llamado **programación reflectiva**.
- Se le llama programación reflectiva al paradigma que busca abstraer el concepto de instrucción a un nivel más alto.
- Teniendo en consideración la programación "normal" se pueden distinguir dos cosas bien diferenciadas: **datos e instrucciones**; los datos se procesan y las instrucciones se ejecutan.

Introducción a Reflection (2)

- ¿Para qué sirve hacer esta distinción?
  - Con Reflection, la invocación a una función puede ser tratada como *operacion.ejecutar*();
  - Se puede generar código para que cualquier clase que cumpla una interfaz determinada, pueda ser instanciada y utilizada sin necesidad de crear una dependencia entre el código cliente y dicha clase.
- ¿Qué permite Reflection?
  - Subir el nivel de abstracción con el cual se programa.
  - Manipular clases, atributos, operaciones, etc., como objetos.
  - Escribir código que pueda ser modificado, actualizado, mejorado, etc. **en tiempo de ejecución**.

Introducción a Reflection (3)

- ¿Para qué sirve hacer esta distinción?
- En pocas palabras: le permite a un programa observar y modificar la estructura y comportamiento de su código dinámicamente.
- Esta API es muy útil para programadores que tienen buenos conocimientos del lenguaje y les permite tener una gran flexibilidad a la hora de desarrollar.
- Las aplicaciones típicas de Reflection son:
  - Manipular clases no conocidas a priori o que serán agregadas de fuentes externas.
  - Visores de clases (que permiten inspeccionar clases).
  - etc.

API de Reflection (1)

- Reflection es un API compleja y extensa, por lo que sólo se verán los puntos necesarios para entender el resto de los conceptos y tener una idea básica de Reflection.
- Se recomienda profundizar en las referencias dadas.
- Las clases de Reflection se encuentran en **java.lang.reflect**
- Típicamente las clases de este paquete no serán instanciadas por el programador sino por invocaciones a alguna operación de la clase Class, contenida en el paquete **java.lang**

API de Reflection (2)

- Clases importantes de la API:
  - **Class**: Representa una clase de la cual se pueden crear instancias, manipularlas, consultar operaciones, atributos, etc.
  - **Method**: Representa un método de una clase que se podrá invocar.
  - **Constructor**: Representa un constructor de una clase al que se le podrá consultar su nombre, el tipo de datos de sus parámetros, etc.
  - **Field**: Representa un atributo de una clase que se podrá consultar y modificar su valor.

API de Reflection (3)

- Se verán 3 formas de obtener un objeto Class:
  - A partir de un String con el nombre y ubicación de la clase:

    Class c = Class.forName("logica.Persona");

    → Se debe manejar la excepción ClassNotFoundException.
  - A partir de un objeto ya existente:

    Persona p = new Persona(…);

    Class c = p.getClass();

    → La operación getClass() pertenece a la clase Object.
  - A partir de un tipo de datos:

    Class c = int.class;

    ó

    Class c = Persona.class;

API de Reflection (4)

- La clase **Class** es la base de Reflection. Algunas operaciones de ***Class***:
  - ***Class Class.forName(String className)*** → Operación estática que retorna el objeto Class asociado al nombre pasado por parámetro (nombre de la clase *fully packaged*). Lanza ***ClassNotFoundException*** si la clase no es encontrada en el CLASSPATH (ya sea como **.class** o dentro de un **.jar**).
  - ***Object newInstance()*** → Crea una nueva instancia del **Class** sobre el que se invoque la operación. Cuidado: se invoca al constructor por defecto de la clase.
  - ***String getName()*** → Retorna el nombre de la clase como un String.

API de Reflection (5)

- Algunas operaciones de ***Class (cont.)***:
  - ***Method[] getMethods()*** → Retorna un array con todos los métodos públicos de la clase.
  - ***Method getMethod(String name, Class param1, ...)*** → Dado un nombre de operación y los tipos de los parámetros (cantidad variable de parámetros) retorna el objeto **Method** correspondiente.
  - ***Constructor[] getConstructors()*** → Retorna un array con los constructores públicos de la clase.
  - ***Constructor getConstructor(Class param1, ...)*** → Retorna el constructor cuyos parámetros coincidan con los indicados como parámetro (cantidad variable de parámetros de tipo **Class**).

API de Reflection (6)

- Algunas operaciones de ***Class (cont.)***:
  - ***Method[] getMethods()*** → Retorna un array con todos los métodos públicos de la clase.
  - ***Field[] getFields()*** → Retorna un array con todos los atributos públicos que tiene la clase sobre la que se invoca la operación.
  - ***Field getField(String name)*** → Retorna un objeto Field dado el nombre del atributo que se requiere.
  - ***boolean isInterface()*** → Retorna true si el objeto Class representa una interfaz y false si representa una clase.
  - ***Class getSuperClass()*** → Retorna un objeto Class representando la superclase del objeto invocante.
  - ***Class[] getInterfaces()*** → Retorna un array con todas las interfaces que implementa la clase.

API de Reflection (7)

- La clase **Method** representa un método. Algunas operaciones de ***Method***:
  - ***Object invoke(Object o, Object arg1, ...)*** → Ejecuta el método con los parámetros que se le pasan al invoke y retorna lo que retorna el método original. La lista de argumentos (valores de parámetros) es variable.
  - ***Class[] getParameterTypes()*** → Retorna un array con los objetos **Class** que representan a los parámetros del método.
  - ***Class getReturnType()*** → Retorna un objeto **Class** que representa al tipo de retorno del método.

API de Reflection (8)

- La clase **Field** representa un atributo. Algunas operaciones de ***Field***:
  - ***String getName()*** → Retorna el nombre del atributo.
  - ***Class getType()*** → Retorna el objeto **Class** asociado al tipo del atributo.
  - ***int getInt(Object o)*** → Retorna un **int** que es el valor de ese atributo en el objeto 'o'. Existen análogas operaciones para los otros tipos de datos primitivos. El atributo debe haber sido declarado como público.
  - ***void setInt(Object o, int valor)*** → Setea el valor del atributo actual al valor pasado como parámetro sobre el objeto 'o'. Existen análogas para los otros tipos de datos primitivos. El atributo debe haber sido declarado como público.

API de Reflection (9)

- Algunas operaciones de ***Field*** (cont):
  - ***Object get(Object o)*** → Retorna un Object que es el valor de ese atributo para el objeto 'o'. Esta operación trabaja con tipos de datos no primitivos (es decir con tipos referencia, como por ejemplo String). El atributo debe haber sido declarado como público.
  - ***void set(Object o, Object valor)*** → Setea el valor del atributo actual al valor pasado como parámetro sobre el objeto 'o'. Esta operación trabaja con tipos de datos no primitivos (es decir con tipos referencia, como por ejemplo String). El atributo debe haber sido declarado como público.