TD N°2 : Réflexivité et introspection

Philippe Lahire

Exercice 1 : Analyse du contenu d'une classe

Ecrire un programme Java auquel on donne le nom d'une classe (une chaîne de caractères) et qui affiche toute l'information que l'on peut obtenir par introspection (modificateurs, classes dont elle hérite, interfaces implémentées, membres avec leur type et leur modificateur).

Vous pouvez si vous le souhaitez partir du squelette de programme qui est mis à votre disposition pour le compléter.

Le programme **`javap`** fournit avec le JDK fait le travail demandé. On s'en servira pour voir le résultat attendu. Par exemple, **`javap java.lang.Class`**.

```
public final class java.lang.Class<T> implements java.io.Serializable, java.lang.reflect.GenericDeclaration, java.lang.reflect.Type, java.lang.reflect.AnnotatedElement {
  transient java.lang.ClassValue$ClassValueMap classValueMap;
  public java.lang.String toString();
  public java.lang.String toGenericString();
  public static java.lang.Class<?> forName(java.lang.String) throws java.lang.ClassNotFoundException;
  public static java.lang.Class<?> forName(java.lang.String, boolean, java.lang.ClassLoader) throws java.lang.ClassNotFoundException;
  public static java.lang.Class<?> forName(java.lang.Module, java.lang.String);
  public T newInstance() throws java.lang.InstantiationException, java.lang.IllegalAccessException;
  public native boolean isInstance(java.lang.Object);
  public native boolean isAssignableFrom(java.lang.Class<?>);
  public native boolean isInterface();
  public native boolean isArray();
  public native boolean isPrimitive();
  public boolean isAnnotation();
  public boolean isSynthetic();
  public java.lang.String getName();
  public java.lang.ClassLoader getClassLoader();
  java.lang.ClassLoader getClassLoader0();
  public java.lang.Module getModule();
  public java.lang.reflect.TypeVariable<java.lang.Class<T>>[] getTypeParameters();
  public native java.lang.Class<? super T> getSuperclass();
  public java.lang.reflect.Type getGenericSuperclass();
  public java.lang.Package getPackage();
  public java.lang.String getPackageName();
  public java.lang.Class<?>[] getInterfaces();
  public java.lang.reflect.Type[] getGenericInterfaces();
  public java.lang.Class<?> getComponentType();
  public native int getModifiers();
  public native java.lang.Object[] getSigners();
  native void setSigners(java.lang.Object[]);
  public java.lang.reflect.Method getEnclosingMethod() throws java.lang.SecurityException;
  public java.lang.reflect.Constructor<?> getEnclosingConstructor() throws java.lang.SecurityException;
  public java.lang.Class<?> getDeclaringClass() throws java.lang.SecurityException;
  public java.lang.Class<?> getEnclosingClass() throws java.lang.SecurityException;
  public java.lang.String getSimpleName();
  public java.lang.String getTypeName();
  public java.lang.String getCanonicalName();
  public boolean isAnonymousClass();
  public boolean isLocalClass();
  public boolean isMemberClass();
  public java.lang.Class<?>[] getClasses();
  public java.lang.reflect.Field[] getFields() throws java.lang.SecurityException;
  public java.lang.reflect.Method[] getMethods() throws java.lang.SecurityException;
  public java.lang.reflect.Constructor<?>[] getConstructors() throws java.lang.SecurityException;
  public java.lang.reflect.Field getField(java.lang.String) throws java.lang.NoSuchFieldException, java.lang.SecurityException;
  public java.lang.reflect.Method getMethod(java.lang.String, java.lang.Class<?>...) throws java.lang.NoSuchMethodException, java.lang.SecurityException;
```

```
 public java.lang.reflect.Constructor<T> getConstructor(java.lang.Class<?>...) throws java.lang.NoSuchMethodException, java.lang.SecurityException;
 public java.lang.Class<?>[] getDeclaredClasses() throws java.lang.SecurityException;
 public java.lang.reflect.Field[] getDeclaredFields() throws java.lang.SecurityException;
 public java.lang.reflect.Method[] getDeclaredMethods() throws java.lang.SecurityException;
 public java.lang.reflect.Constructor<?>[] getDeclaredConstructors() throws java.lang.SecurityException;
 public java.lang.reflect.Field getDeclaredField(java.lang.String) throws java.lang.NoSuchFieldException, java.lang.SecurityException;
 public java.lang.reflect.Method getDeclaredMethod(java.lang.String, java.lang.Class<?>...) throws java.lang.NoSuchMethodException, java.lang.SecurityException;
 java.util.List<java.lang.reflect.Method> getDeclaredPublicMethods(java.lang.String, java.lang.Class<?>...);
 public java.lang.reflect.Constructor<T> getDeclaredConstructor(java.lang.Class<?>...) throws java.lang.NoSuchMethodException, java.lang.SecurityException;
 public java.io.InputStream getResourceAsStream(java.lang.String);
 public java.net.URL getResource(java.lang.String);
 public java.security.ProtectionDomain getProtectionDomain();
 static native java.lang.Class<?> getPrimitiveClass(java.lang.String);
 native byte[] getRawAnnotations();
 native byte[] getRawTypeAnnotations();
 static byte[] getExecutableTypeAnnotationBytes(java.lang.reflect.Executable);
 native jdk.internal.reflect.ConstantPool getConstantPool();
 public boolean desiredAssertionStatus();
 public boolean isEnum();
 public T[] getEnumConstants();
 T[] getEnumConstantsShared();
 java.util.Map<java.lang.String, T> enumConstantDirectory();
 public T cast(java.lang.Object);
 public <U> java.lang.Class<? extends U> asSubclass(java.lang.Class<U>);
 public <A extends java.lang.annotation.Annotation> A getAnnotation(java.lang.Class<A>);
 public boolean isAnnotationPresent(java.lang.Class<? extends java.lang.annotation.Annotation>);
 public <A extends java.lang.annotation.Annotation> A[] getAnnotationsByType(java.lang.Class<A>);
 public java.lang.annotation.Annotation[] getAnnotations();
 public <A extends java.lang.annotation.Annotation> A getDeclaredAnnotation(java.lang.Class<A>);
 public <A extends java.lang.annotation.Annotation> A[] getDeclaredAnnotationsByType(java.lang.Class<A>);
 public java.lang.annotation.Annotation[] getDeclaredAnnotations();
 boolean casAnnotationType(sun.reflect.annotation.AnnotationType, sun.reflect.annotation.AnnotationType);
 sun.reflect.annotation.AnnotationType getAnnotationType();
 java.util.Map<java.lang.Class<? extends java.lang.annotation.Annotation>, java.lang.annotation.Annotation> getDeclaredAnnotationMap();
 public java.lang.reflect.AnnotatedType getAnnotatedSuperclass();
 public java.lang.reflect.AnnotatedType[] getAnnotatedInterfaces();
 public java.lang.Class<?> getNestHost();
 public boolean isNestmateOf(java.lang.Class<?>);
 public java.lang.Class<?>[] getNestMembers();
 static {};
}
```

Note : ne pas oublier d'inclure les classes imbriquées (classes membres statiques ou non uniquement car vous n'avez pas accès aux classes locales, anonymes ou les lambda).

Exercice 2 : Réaliser une méthode toString() générique

Utiliser l'introspection pour créer un méthode **`toString()`** générique. Il s'agit en pratique de faire une méthode **`toString`** qui prend en paramètre un objet de type **`Object`** et affiche la valeur de chacun de ses champs. Attention, si les champs sont des références sur d'autres objets on descendra en profondeur pour afficher "récursivement" leur valeur également.

Pour permettre l'arrêt du programme, la méthode **`toString`** aura un deuxième paramètre qui est la profondeur à laquelle on souhaite descendre.