Core Java

Agenda

- Reflection
- Annotations

Reflection

- .class = Byte-code + Meta-data + Constant pool + ...
- When class is loaded into JVM all the metadata is stored in the object of java.lang.Class (heap area).
- This metadata includes class name, super class, super interfaces, fields (field name, field type, access modifier, flags), methods (method name, method return type, access modifier, flags, method arguments, ...), constructors (access modifier, flags, ctor arguments, ...), annotations (on class, fields, methods, ...).

Reflection applications

- Inspect the metadata (like javap)
- Build IDE/tools (Intellisense)
- Dynamically creating objects and invoking methods
- Access the private members of the class

Get the java.lang.Class object

- way 1: When you have class-name as a String (taken from user or in properties file)

```
Class<?> c = Class.forName(className);
```

- way 2: When the class is in project/classpath.

```

```

```
Class<?> c = ClassName.class;
```

- way 3: When you have object of the class.

```
Class<?> c = obj.getClass();
```

Access metadata in java.lang.Class

- Name of the class

```
String name = c.getName();
```

- Super class of the class

```
Class<?> supcls = c.getSuperclass();
```

- Super interfaces of the class

```
Class<?> supintf[] = c.getInterfaces();
```

- Fields of the class

```
Field[] fields = c.getFields(); // all fields accessible (of class & its super class)
```

```
Field[] fields = c.getDeclaredFields(); // all fields in the class
```

- Methods of the class

```
Method[] methods = c.getMethods(); // all methods accessible (of class & its super class)
```

```
Method[] methods = c.getDeclaredMethods(); // all methods in the class
```

- Constructors of the class

```
Constructor[] ctors = c.getConstructors(); // all ctors accessible (of class & its super class)
```

```
Constructor[] ctors = c.getDeclaredConstructor(); // all ctors in the class
```

Invoking method dyanmically

```
```Java
public class Middleware {
    public static Object invoke(String className, String methodName, Class[] methodParamTypes, Object[] methodArgs) throws
Exception {
        // load the given class
        Class c = Class.forName(className);
        // create object of that class
```
```

```
        Object obj = c.newInstance(); // also invokes param-less constructor
        // find the desired method
        Method method = c.getDeclaredMethod(methodName, methodParamTypes);
        // allow to access the method (irrespective of its access specifier)
        method.setAccessible(true);
        // invoke the method on the created object with given args & collect the result
        Object result = method.invoke(obj, methodArgs);
        // return the results
        return result;
    }
}
```
```Java
// invoking method statically
Date d = new Date();
String result = d.toString();
```
```Java
// invoking method dyanmically
String result = Middleware.invoke("java.util.Date", "toString", null, null);
```

Reflection Tutorial

- https://youtu.be/lAoNJ_7LD44
- https://youtu.be/UVWdtk5ibK8

Annotations

- Added in Java 5.0.
- Annotation is a way to associate metadata with the class and/or its members.
- Annotation applications
    - Information to the compiler

    - Compile-time/Deploy-time processing
    - Runtime processing
- Annotation Types
    - Marker Annotation: Annotation is not having any attributes.
        - @Override, @Deprecated, @FunctionalInterface ...
    - Single value Annotation: Annotation is having single attribute -- usually it is "value".
        - @SuppressWarnings("deprecation"), ...
    - Multi value Annotation: Annotation is having multiple attribute
        - @RequestMapping(method = "GET", value = "/books"), ...

Pre-defined Annotations

- @Override
    - Ask compiler to check if corresponding method (with same signature) is present in super class.
    - If not present, raise compiler error.
- @FunctionalInterface
    - Ask compiler to check if interface contains single abstract method.
    - If zero or multiple abstract methods, raise compiler error.
- @Deprecated
    - Inform compiler to give a warning when the deprecated type/member is used.
- @SuppressWarnings
    - Inform compiler not to give certain warnings: e.g. deprecation, rawtypes, unchecked, serial, unused
    - @SuppressWarnings("deprecation")
    - @SuppressWarnings({"rawtypes", "unchecked"})
    - @SuppressWarnings("serial")
    - @SuppressWarnings("unused")

Meta-Annotations

- Annotations that apply to other annotations are called meta-annotations.
- Meta-annotation types defined in java.lang.annotation package.

@Retention

- RetentionPolicy.SOURCE
  - Annotation is available only in source code and discarded by the compiler (like comments).
  - Not added into .class file.
  - Used to give information to the compiler.
  - e.g. @Override, ...
- RetentionPolicy.CLASS
  - Annotation is compiled and added into .class file.
  - Discared while class loading and not loaded into JVM memory.
  - Used for utilities that process .class files.
  - e.g. Obfuscation utilities can be informed not to change the name of certain class/member using @SerializedName, ...
- RetentionPolicy.RUNTIME
  - Annotation is compiled and added into .class file. Also loaded into JVM at runtime and available for reflective access.
  - Used by many Java frameworks.
  - e.g. @RequestMapping, @Id, @Table, @Controller, ...

@Target

- Where this annotation can be used.
- ANNOTATION_TYPE, CONSTRUCTOR, FIELD, LOCAL_VARIABLE, METHOD, PACKAGE, PARAMETER, TYPE, TYPE_PARAMETER, TYPE_USE
- If annotation is used on the other places than mentioned in @Target, then compiler raise error.

@Documented

- This annotation should be documented by javadoc or similar utilities.

@Repeatable

- The annotation can be repeated multiple times on the same class/target.

@Inherited

- The annotation gets inherited to the sub-class and accessible using c.getAnnotation() method.

Custom Annotation

- Annotation to associate developer information with the class and its members.

```
@Inherited
@Retention(RetentionPolicy.RUNTIME) // the def attribute is considered as "value" = @Retention(value =
RetentionPolicy.RUNTIME )
@Taget({TYPE, CONSTRUCTOR, FIELD, METHOD}) // { } represents array
@interface Developer {
    String firstName();
    String lastName();
    String company() default "Sunbeam";
    String value() default "Software Engg";
}

@Repeatable
@Retention(RetentionPolicy.RUNTIME)
@Taget({TYPE})
@interface CodeType {
    String[] value();
}
```

```
//@Developer(firstName="Nilesh", lastName="Ghule", value="Technical Director") // compiler error -- @Developer is not
@Repeatable
@CodeType({"businessLogic", "algorithm"})
@Developer(firstName="Nilesh", lastName="Ghule", value="Technical Director")
class MyClass {
    // ...
    @Developer(firstName="Shubham", lastName="Borle", company="Sunbeam Karad ")
    private int myField;
    @Developer(firstName="Rahul", lastName="Sansuddi")
```

```
    public MyClass() {

    }
    @Developer(firstName="Shubham", lastName="Borle", company="Sunbeam Karad ")
    public void myMethod() {
        @Developer(firstName="James", lastName="Bond") // compiler error
        int localVar = 1;
    }
}
```

```
// @Developer is inherited
@CodeType("frontEnd")
@CodeType("businessLogic") // allowed because @CodeType is @Repeatable
class YourClass extends MyClass {
    // ...
}
```

Annotation processing (using Reflection)

```
Annotation[] anns = MyClass.class.getDeclaredAnnotations();
for (Annotation ann : anns) {
    System.out.println(ann.toString());
    if(ann instanceof Developer) {
        Developer devAnn = (Developer) ann;
        System.out.println(" - Name: " + devAnn.firstName() + " " + devAnn. lastName());
        System.out.println(" - Company: " + devAnn.company());
        System.out.println(" - Role: " + devAnn.value());
    }
}
System.out.println();

Field field = MyClass.class.getDeclaredField("myField");
```

```
anns = field.getAnnotations() ;
for (Annotation ann : anns)
    System.out.println(ann.toString());
System.out.println();

//anns = YourClass.class.getDeclaredAnnotations();
anns = YourClass.class.getAnnotations();
for (Annotation ann : anns)
    System.out.println(ann.toString());
System.out.println();
```

Annotation tutorials

- Part 1: https://youtu.be/7zjWPJqlPRY
- Part 2: https://youtu.be/CafN2ABJQcg

Java Proxies (Tutorials)

- Not in our/C-DAC syllabus, but a useful topic.
- Part 1: https://youtu.be/4X_sZNOeR7g
- Part 2: https://youtu.be/jRv3GJuaudA