Core Java

Day17 - Agenda

- Reflection
- Annotation
- Java IO Framework

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 tutorials

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

Java IO framework

- Input/Output functionality in Java is provided under package java.io and java.nio package.

- IO framework is used for File IO, Network IO, Memory IO, and more.
- File is a collection of data and information on a storage device.
- File = Data + Metadata
- Two types of APIs are available file handling
  - FileSystem API -- Accessing/Manipulating Metadata
  - File IO API -- Accessing/Manipulating Contents/Data

java.io.File class

- A path (of file or directory) in file system is represented by "File" object.
- Used to access/manipulate metadata of the file/directory.
- Provides FileSystem APIs
  - String[] list() -- return contents of the directory
  - File[] listFiles() -- return contents of the directory
  - boolean exists() -- check if given path exists
  - boolean mkdir() -- create directory
  - boolean mkdirs() -- create directories (child + parents)
  - boolean createNewFile() -- create empty file
  - boolean delete() -- delete file/directory
  - boolean renameTo(File dest) -- rename file/directory
  - String getAbsolutePath() -- returns full path (drive:/folder/folder/...)
  - String getPath() -- return path
  - File getParentFile() -- returns parent directory of the file
  - String getParent() -- returns parent directory path of the file
  - String getName() -- return name of the file/directory
  - static File[] listRoots() -- returns all drives in the systems.
  - long getTotalSpace() -- returns total space of current drive
  - long getFreeSpace() -- returns free space of current drive
  - long getUsableSpace() -- returns usable space of current drive
  - boolean isDirectory() -- return true if it is a directory
  - boolean isFile() -- return true if it is a file
  - boolean isHidden() -- return true if the file is hidden
  - boolean canExecute()
  - boolean canRead()
  - boolean canWrite()
  - boolean setExecutable(boolean executable) -- make the file executable
  - boolean setReadable(boolean readable) -- make the file readable
  - boolean setWritable(boolean writable) -- make the file writable
  - long length() -- return size of the file in bytes
  - long lastModified() -- last modified time
  - boolean setLastModified(long time) -- change last modified time

Java IO

- Java File IO is done with Java IO streams.
- Stream is abstraction of data source/sink.
  - Data source -- InputStream or Reader

  - Data sink -- OutputStream or Writer
- Java supports two types of IO streams.
  - Byte streams (binary files) -- byte by byte read/write
  - Character streams (text files) -- char by char read/write
- All these streams are AutoCloseable (so can be used with try-with-resource construct)