The purpose of this assignment is to write and use the annotations in Java.

Exercise 1 – CodeGen annotations

Open the `CodeGen/` folder contents in your IDE. The goal of this exercise is to implement an *@Extract* annotation and write a program that *encapsulates* the given *Target* class, by generating getters and setters for its **annotated with @Extract** fields (public, private, protected or package-private). The annotation should have a value field: when a value is passed, the annotated field should be renamed in the output class.
The resulting source code of the encapsulated class should be printed to the console.
While implementing the solution consider that:

1) All fields should be converted to private fields.
2) camelCase naming convention should be used for generating the getters and setters.
3) The *Extract* annotation should throw an exception if the proposed field name is already in use (you cannot redeclare a field with the same name)

Example:

Target class

```
public class Target {
  @Extract
  public int theAnswer = 42;

  @Extract(“foo”)
  private String hello = "world";
}
```

Desired console output

```
public class Target {
  private int theAnswer = 42;
  private String foo = "world";

  public int getTheAnswer() {
    return theAnswer;
  }

  public String getFoo() {
    return foo;
  }

  public void setTheAnswer(int theAnswer) {
    this.theAnswer = theAnswer;
  }

  public void setFoo(String foo) {
    this.foo = foo;
  }
}
```

Exercise 2 – Markdown annotations

Open the `markdown/` folder contents in your IDE. It contains an example *Coordinate* class, the *MarkdownDoc* / *MarkdownDocIgnore* interfaces and *MarkdownGenerator* class which contains the skeleton of the main class.
The goal is to declare the *@MarkdownDoc* and *@MarkdownDocIgnore* annotations (used by *Coordinate.java*) by modifying the provided interfaces and then to implement the *MarkdownGenerator* class to read those annotations, using reflection. The markdown documentation structure, produced by the generator, should be printed to console.
Consider the following indications when implementing the annotations:

- **MarkdownDoc** can be used only on classes and identifies the classes that should be documented. It must also allow to establish, which elements should be documented (parent class, interfaces, fields, costructors and methods) by providing convenient boolean element declarations, all defaulting to true.
- **MarkdownDocIgnore** can be used only on fields, constructors and methods and marks elements that should not be document.

Example output:

```
# Class `assignment05.markdown.Coordinate`
```

```
Parent class: `java.lang.Object`
## Interface(s)
 - `java.lang.Comparable`
 - `java.io.Serializable`
## Fields(s)
 - `float lat`
 - `float lon`
## Constructor(s)
 - `assignment05.markdown.Coordinate(float, float)`
## Methods(s)
 - `float getLon()`
 - `boolean isValidLat(float)`
 - `float getLat()`
 - `boolean isValidLon(float)`
 - `int compareTo(java.lang.Object)`
 - `int compareTo(assignment05.markdown.Coordinate)`
 - `double distance(assignment05.markdown.Coordinate)`
```

Exercise 3 – Validation (Optional)

Open the `validation/` folder contents in your IDE. It contains the starting interfaces to implement the *@Validate* / *@ValidationItem* annotations and the example *MathOperations* utility class that uses the annotations.
The *@ValidationItem* annotation provides the input parameters and the expected result used to validate the method's implementation.
The *@Validate* annotation contains one or more *@ValidationItem* annotations and a verbosity field, meant to customize the verbosity level of the validation set. The verbosity parameter should be an enum (TRACE, ERRORS_ONLY) and the default value is ERRORS_ONLY.

The goal of this exercise is to implement the annotation interfaces, without changing the *MathOperations* class. It is **not required** to read the annotations and compute the actual validation of the methods!