Example 1

- Create a simple unit-test framework based on annotations (ala JUnit)
- Supported annotations
  - `@Test`
    - marking a test method
      - if the method terminates without an exception -> Test OK
      - if terminates with an exception -> Test fails
    - two option arguments
      - boolean enabled – default true
        - if false, the test is ignored
      - `Class[]` expectedExceptions – default an empty array
        - if the test terminates with an exception the array, it is OK
  - `@Before`
    - the methods is executed before each test in the given class
  - `@After`
    - the methods is executed after each test in the given class
Example 1

• Supported annotations (cnt.)
  – @TesterInfo
    • only for classes
    • arguments
      – String createdBy
      – String lastModified
      – Priority priority
        (enum Priority {HIGH,MEDIUM,LOW})
Example 1

- An example of usage

```java
@TestInfo(priority = Priority.HIGH, createdBy = "PH", lastModified = "01/01/2018")
public class SimpleTest {
    private Collection<String> collection;

    @Before
    public void setUp() {
        collection = new ArrayList<>();
    }

    @After
    public void tearDown() {
        collection.clear();
    }

    @Test
    public void testEmptyCollection() {
        if (!collection.isEmpty()) throw new AssertionError();
    }

    @Test
    public void testOneItemCollection() {
        collection.add("itemA");
        if (1 != collection.size()) throw new AssertionError();
    }
}
```
Example 1

• Write the annotations and a launcher
  – the launcher gets a file with a list of classes to be searched for tests
  – for each class, the launcher prints out information from @TesterInfo and for each test, whether it passed or failed
  – classes are processed (and printed out) by the priority in @TesterInfo
  – after all classes are processed, overall statistics are printed out, e.g.

  Result: Total 3, Passed 1, Failed 1, Ignore 1