
approval Documentation

Release 0.2

nikolavp

Sep 27, 2017

Contents

1	What can it be used for?	3
----------	---------------------------------	----------

Approval provides a powerful toolkit of ways to test the behavior of critical components so you can prevent problems in your production environment.

CHAPTER 1

What can it be used for?

Approval can be used for verifying objects that require more than a simple assert. The idea is that you sometimes just want to verify a particular result at the end and then start implementation refactoring. I like to call it “I will know the right result when I see it”. Usecases for this might be:

- performance improvements to the implementation while preserving the current system output
- just verifying RESTful response results, be it JSON, XML, HTML whatever
- people use it for TDD but instead of providing the result upfront you just implement the simple possible thing, verify the result and then start improving the implementation.

Getting Started

Getting Started will guide you through the process of testing your classes with approval testing. Don't worry if you are used to normal testing with assertions you will get up to speed in minutes.

Setting Up Maven

Just add the `approval` library as a dependency:

```
<dependencies>
  <dependency>
    <groupId>com.github.nikolavp</groupId>
    <artifactId>approval</artifactId>
    <version>${approval.version}</version>
  </dependency>
</dependencies>
```

Warning: Make sure you have a `approval.version` property declared in your POM with the current version, which is 0.2.

How is approval testing different

There are many sources from which you can learn about approval testing(just google it) but basically the process is the following:

1. you already have a working implementation of the thing you want to test
2. you run it and get the result the first time
3. the result will be shown to you in your preferred tool(this can be configured)
4. you either approve the result in which case it is recored(saved) and the test pass or you disapprove it in which case the test fails
5. the recorded result is then used on further test runs to make sure that there are no regressions in your code(i.e. you broke something and the result is not the same).
6. Of course sometimes you want to change the way something behaves so if the result is not the same we will prompt you with difference between the new result and the last recorded again in your preferred tool.

Approvals utility

This is the main starting point of the library. If you want to just approve a primitive object or arrays of primitive object then you are ready to go. The following will start the approval process for a `String` that `MyCoolThing` (our class under test) generated and use `src/test/resources/approval/string.verified` for recording/saving the results:

```
@Test
public void testMyCoolThingReturnsProperString() {
    String result = MyCoolThing.getComplexMultilineString();
    Approvals.verify(result, Paths.get("src", "resources", "approval", "result.txt
↪"));
}
```

Approval class

This is the main object for starting the approval process. Basically it is used like this:

```
@Test
public void testMyCoolThingReturnsProperStringControlled() {
    String string = MyCoolThing.getComplexMultilineString();
    Approval<String> approver = Approval.of(String.class)
        .withReporter(Reporters.console())
        .build();
    approver.verify(string, Paths.get("src", "resources", "approval", "string.
↪verified"));
}
```

note how this is different from [Approvals utility](#) - we are building a custom `Approval` object which allows us to control and change the whole approval process. Look at [Reporter class](#) and [Converter](#) for more info.

Note: `Approval` object are thread safe so you are allowed to declare them as static variables and reuse them in all your tests. In the example above if we have more testing methods we can only declare the `Approval` object once as a static variable in the `Test` class

Reporter class

Reporters(in lack of better name) are used to prompt the user for approving the result that was given to the *Approval* object. There is a *withReporter* method on *ApprovalBuilder* that allows you to use a custom reporter. We provide some ready to use reporters in the following classes:

- *Reporters* - this factory class contains cross platform programs/reporters. Here you will find things like *gvim*, *console*.
- *WindowsReporters* - this factory class contains Windows programs/reporters. Here you will find things like *notepadPlusPlus*, *beyondCompare*, *tortoiseText*.
- *MacOSReporters* - this factory class contains MacOS programs/reporters. Here you will find things like *diffMerge*, *ksdiff*, etc.

Note: Sadly I am unable to properly test the windows and macOS reporters because I mostly have access to Linux machines. If you find a problem, blame it on me.

Converter

Converters are objects that are responsible for serializing objects to raw form(currently `byte[]`). This interface allows you to create a custom converter for your custom objects and reuse the approval process in the library. We have converters for all primitive types, String and their array variants. Of course providing a converter for your custom object is dead easy. Let's say you have a custom entity class that you are going to use for verifications in your tests:

```
package com.nikolavp.approval.example;

public class Entity {

    private String name;
    private int age;

    public Entity(String name, int age) {
        this.age = age;
        this.name = name;
    }

    public String getName() {
        return name;
    }

    public int getAge() {
        return age;
    }

}
```

Here is a possible simple converter for the class:

```
package com.nikolavp.approval.example;

import com.nikolavp.approval.converters.Converter;

import javax.annotation.Nonnull;
import java.nio.charset.StandardCharsets;
```

```
public class EntityConverter implements Converter<Entity> {
    @Nonnull
    @Override
    public byte[] getRawForm(Entity value) {
        return ("Entity is:\n" +
            "age = " + value.getAge() + "\n" +
            "name = " + value.getName() + "\n").getBytes(StandardCharsets.UTF_8);
    }
}
```

now let's say we execute a simple test

```
Entity entity = new Entity("Nikola", 30);
Approval<Entity> approver = Approval.of(Entity.class)
    .withReporter(Reporters.console())
    .withConverter(new EntityConverter())
    .build();
approver.verify(entity, Paths.get("src/test/resources/approval/example/entity.
↪verified"));
}
```

we will get the following output in the console(because we are using the console reporter)

```
Entity is:
age = 30
name = Nikola
```

Path Mapper

Path mapper are used to abstract the way in which the final path file that contains the verification result is built. You are not required to use them but if you want to add structure to the your approval files you will at some point find the need for them. Let's see an example:

You have the following class containing two verifications:

```
package com.nikolavp.approval.example;

import com.nikolavp.approval.Approval;
import com.nikolavp.approval.reporters.Reporters;
import org.junit.Test;

import java.nio.file.Paths;

public class PathMappersExample {
    private static final Approval<String> APPROVER = Approval.of(String.class)
        .withReporter(Reporters.console())
        .build();

    @Test
    public void shoulProperlyTestString() throws Exception {
        APPROVER.verify("First string test", Paths.get("src", "test", "resources",
↪"approvals", "first-test.txt"));
    }

    @Test
    public void shoulProperlyTestStringSecond() throws Exception {
```

```

        APPROVER.verify("Second string test", Paths.get("src", "test", "resources",
↪ "approvals", "second-test.txt"));
    }
}

```

now if you want to add another approval test you will need to write the same destination directory for the approval path again. You can of course write a private static method that does the mapping for you but we can do better with PathMappers:

```

package com.nikolavp.approval.example;

import com.nikolavp.approval.Approval;
import com.nikolavp.approval.pathmappers.ParentPathMapper;
import com.nikolavp.approval.reporters.Reporters;
import org.junit.Test;

import java.nio.file.Paths;

public class PathMappersExampleImproved {
    private static final Approval<String> APPROVER = Approval.of(String.class)
        .withReporter(Reporters.console())
        .withPathMapper(new ParentPathMapper<String>(Paths.get("src", "test",
↪ "resources", "approvals")))
        .build();

    @Test
    public void shoulProperlyTestString() throws Exception {
        APPROVER.verify("First string test", Paths.get("first-test.txt"));
    }

    @Test
    public void shoulProperlyTestStringSecond() throws Exception {
        APPROVER.verify("Second string test", Paths.get("second-test.txt"));
    }
}

```

we abstracted the common parent directory with the help of the *ParentPathMapper* class. We provide other path mapper as part of the library that you can use:

- *JunitPathMapper*

Limitations

Some things that you have to keep in mind when using the library:

- unordered objects like *HashSet*, *HashMap* cannot be deterministically verified because their representation will vary from run to run.

User Manual

Simple example of the library

Let's try to test the simplest example possible:

```
package com.nikolavp.approval.example;

public class SimpleExample {
    public static String generateHtml(String pageTitle) {
        return String.format(
            "<!DOCTYPE html>\n" +
            "<html lang=\"en\">\n" +
            "<head>\n" +
            "    <title>%s</title>\n" +
            "<meta charset=\"utf-8\"/>\n" +
            "<link href=\"css/myscript.css\" \n" +
            "    rel=\"stylesheet\"/>\n" +
            "<script src=\"scripts/myscript.js\">\n" +
            "</script>\n" +
            "</head>\n" +
            "<body>\n" +
            "... \n" +
            "</body>\n" +
            "</html>", pageTitle);
    }
}
```

now this class is not rocket science and if we want to test `getResult()`, we would write something like the following in JUnit:

```
package com.nikolavp.approval.example;

import org.junit.Test;

import static org.hamcrest.CoreMatchers.equalTo;
import static org.junit.Assert.assertThat;

public class SimpleExampleTest {

    @Test
    public void shouldReturnSomethingToTestOut() throws Exception {
        //arrange
        String title = "myTitle";
        String expected = "<!DOCTYPE html>\n" +
            "<html lang=\"en\">\n" +
            "<head>\n" +
            "    <title>" + title + "</title>\n" +
            "<meta charset=\"utf-8\"/>\n" +
            "<link href=\"css/myscript.css\" \n" +
            "    rel=\"stylesheet\"/>\n" +
            "<script src=\"scripts/myscript.js\">\n" +
            "</script>\n" +
            "</head>\n" +
            "<body>\n" +
            "... \n" +
            "</body>\n" +
            "</html>";

        //act
        String actual = SimpleExample.generateHtml(title);

        //assert
        assertThat(actual, equalTo(expected));
    }
}
```

```
}  
}
```

this is quite terse and short. Can we do better? Actually because we support strings out of the box, approval is a lot shorter

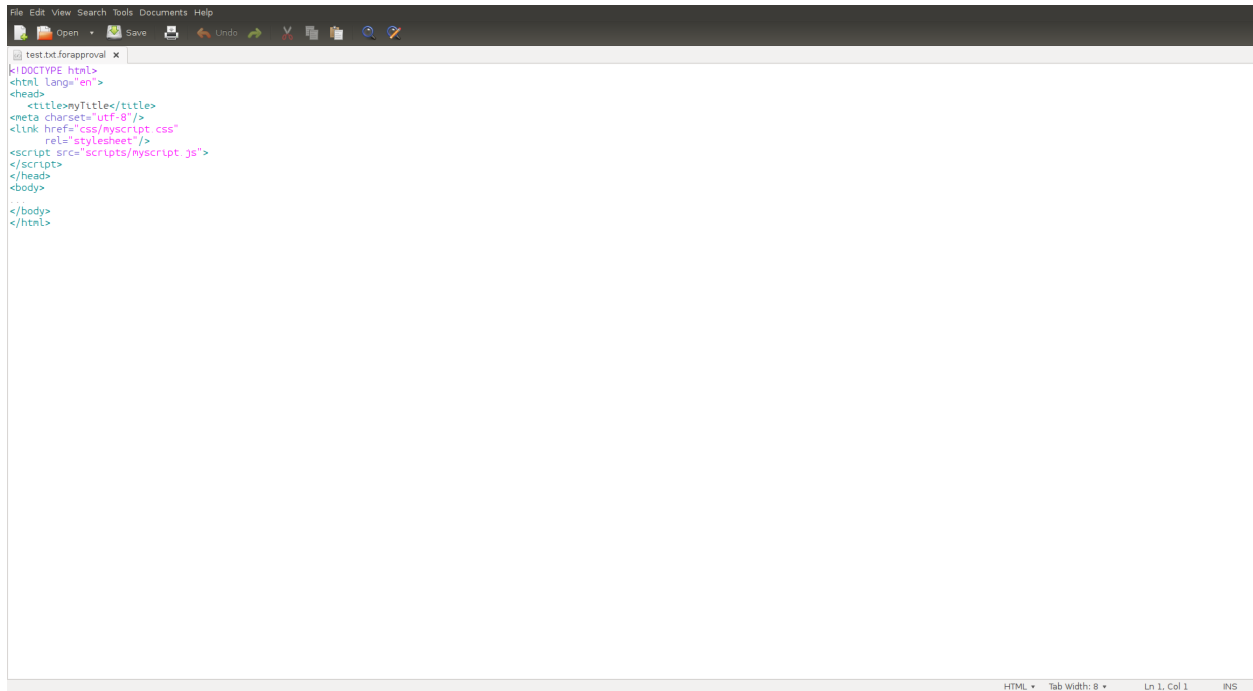
```
package com.nikolavp.approval.example;  
  
import com.nikolavp.approval.Approvals;  
import org.junit.Test;  
  
import java.nio.file.Paths;  
  
public class SimpleExampleApprovalTest {  
  
    @Test  
    public void shouldReturnSomethingToTestOut() throws Exception {  
        //assign  
        String title = "myTitle";  
  
        //act  
        String actual = SimpleExample.generateHtml(title);  
  
        //verify  
        Approvals.verify(actual, Paths.get("test.txt"));  
    }  
}
```

when the latter is executed you will be prompted in your tool of choice to verify the result from **getResult()**. Verifying the result will vary from your tool of choice because some of them allow you to control the resulting file and others just show you what was the verification object.

To see it in action we will look at two possible reporters:

Gedit

Gedit is just a simple editor. When we run the test it will show us the string representation:

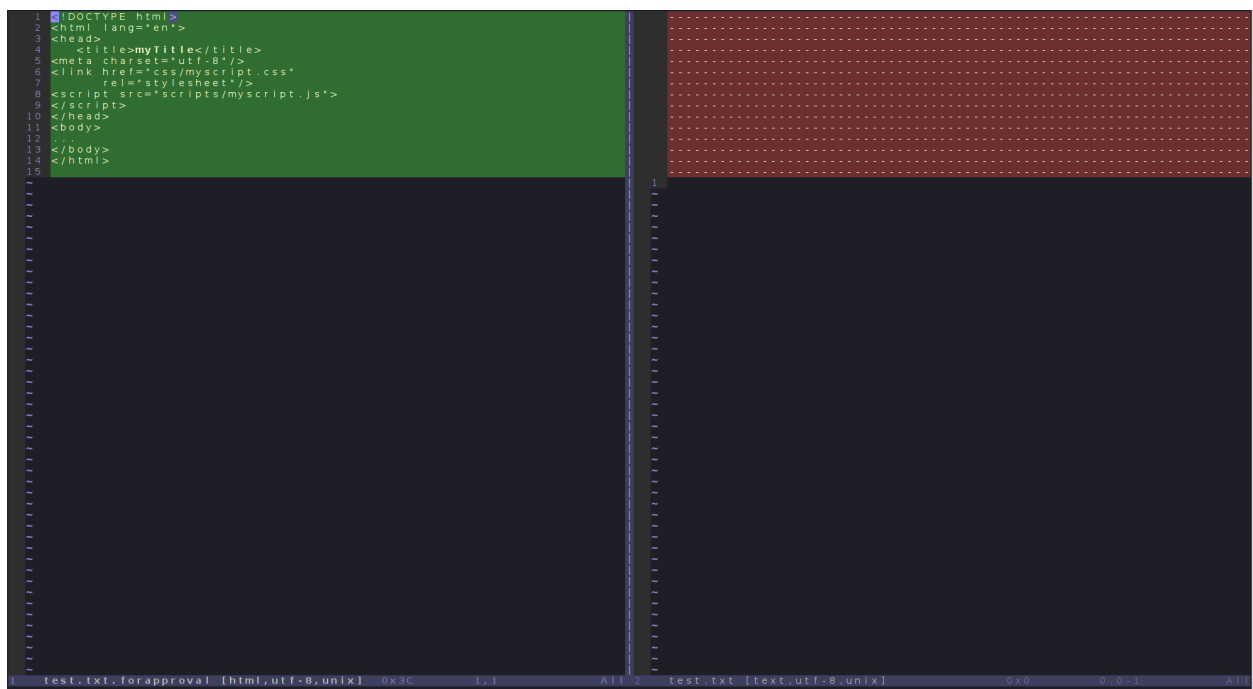


```
File Edit View Search Tools Documents Help
test.txt:forapproval x
<!DOCTYPE html>
<html lang="en">
<head>
  <title>myTitle</title>
  <meta charset="utf-8"/>
  <link href="css/myscript.css"
        rel="stylesheet"/>
  <script src="scripts/myscript.js">
  </script>
</head>
<body>
</body>
</html>
```

as you can see this is the string representation of the result opened in gedit. If we close gedit we will prompted by a confirm window which will ask us if we approve the result or it is not OK. On not OK the test will fail with an `AssertionError` and otherwise the test will pass and will continue to pass until the returned value from `getResult()` changes.

GvimDiff

Gvimdiff is much more powerful than gedit. If we decide to use it then we got the power in our hands and we can decide if we want the file or not(there will be no confirmation window). Here is how it looks like:



```
1 |!DOCTYPE html|
2 |<html lang="en"|
3 |<head>
4 |  <title>myTitle</title>
5 |  <meta charset="utf-8"/>
6 |  <link href="css/myscript.css"
7 |        rel="stylesheet"/>
8 |  <script src="scripts/myscript.js">
9 |  </script>
10 |</head>
11 |<body>
12 |
13 |</body>
14 |</html>
15 |
```

as you can see on the left side is the result from the test run and on the right side is what will be written for consecutive test runs. If we are ok with the result we can get everything from the left side, save the right side and exit vim. The test will now pass and will continue to pass until the returned value from **getResult()** changes.

Let's say someone changes the code and it no longer contains a DOCTYPE declaration. The reporter will fire up and we will get the following window:

```
1 <html lang="en">
2 <head>
3   <title>myTitle</title>
4   <meta charset="utf-8"/>
5   <link href="css/myscript.css"
6     rel="stylesheet"/>
7 +:: B lines: <script src="scripts/myscript.js">:::
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
10
```

we can approve the change or exit our tool and mark the result as invalid.

FAQ

Illegal state exception with “<myclass> is not a primitive type class!”?

This means that you are trying to create/use an *Approval* object that's for a non primitive type and you haven't specified a *Converter*

Javadoc

com.nikolavp.approval

Approval

```
public class Approval<T>
```

The main entry point class for each approval process. This is the main service class that is doing the hard work - it calls other classes for custom logic based on the object that is approved. Created by nikolayp on 1/29/14.

Parameters

- **<T>** – the type of the object that will be approved by this *Approval*

Constructors

Approval

Approval (*Reporter* reporter, *Converter*<T> converter, *PathMapper*<T> pathMapper)

Create a new object that will be able to approve “things” for you.

Parameters

- **reporter** – a reporter that will be notified as needed for approval events
- **converter** – a converter that will be responsible for converting the type for approval to raw form
- **pathMapper** – the path mapper that will be used

Approval

Approval (*Reporter* reporter, *Converter*<T> converter, *PathMapper*<T> pathMapper,
com.nikolavp.approval.utils.FileSystemUtils fileSystemReadWriter)

This ctor is for testing only.

Methods

getApprovalPath

public static *Path* **getApprovalPath** (*Path* filePath)

Get the path for approval from the original file path.

Parameters

- **filePath** – the original path to value

Returns the path for approval

getConverter

Converter<T> **getConverter** ()

getPathMapper

PathMapper<T> **getPathMapper** ()

getReporter

Reporter **getReporter** ()

of

public static <T> *ApprovalBuilder*<T> **of** (*Class*<T> *clazz*)

Create a new approval builder that will be able to approve objects from the specified class type.

Parameters

- **clazz** – the class object for the things you will be approving
- **<T>** – the type of the objects you will be approving

Returns an approval builder that will be able to construct an *Approval* for your objects

verify

public void **verify** (T *value*, *Path* *filePath*)

Verify the value that was passed in.

Parameters

- **value** – the value object to be approved
- **filePath** – the path where the value will be kept for further approval

Approval.ApprovalBuilder

public static final class **ApprovalBuilder**<T>

A builder class for approvals. This is used to conveniently build new approvals for a specific type with custom reporters, converters, etc.

Parameters

- **<T>** – the type that will be approved by the the resulting approval object

Methods

build

public *Approval*<T> **build** ()

Creates a new approval with configuration/options(reporters, converters, etc) that were set for this builder.

Returns a new approval for the specified type with custom configuration if any

withConverter

public *ApprovalBuilder*<T> **withConverter** (*Converter*<T> *converterToBeUsed*)

Set the converter that will be used when building new approvals with this builder.

Parameters

- **converterToBeUsed** – the converter that will be used from the approval that will be built

Returns the same builder for chaining

See also: *Converter*

withPathMapper

public *ApprovalBuilder*<T> **withPathMapper** (*PathMapper*<T> *pathMapperToBeUsed*)

Set a path mapper that will be used when building the path for approval results.

Parameters

- **pathMapperToBeUsed** – the path mapper

Returns the same builder for chaining

withReporter

public *ApprovalBuilder*<T> **withReporter** (*Reporter* *reporterToBeUsed*)

Set the reporter that will be used when building new approvals with this builder.

Parameters

- **reporterToBeUsed** – the reporter that will be used from the approval that will be built

Returns the same builder for chaining

See also: *Reporter*

Approvals

public final class **Approvals**

Approvals for primitive types. This is a convenient static utility class that is the first thing to try when you want to use the library. If you happen to be lucky and need to verify only primitive types or array of primitive types then we got you covered.

User: nikolavp (Nikola Petrov) Date: 07/04/14 Time: 11:38

Methods

verify

public static void **verify** (int[] *ints*, *Path* *path*)

An overload for verifying int arrays. This will call the approval object with proper reporter and use the path for verification.

Parameters

- **ints** – the int array that needs to be verified
- **path** – the path in which to store the approval file

verify

public static void **verify** (byte[] *bytes*, *Path* *path*)

An overload for verifying byte arrays. This will call the approval object with proper reporter and use the path for verification.

Parameters

- **bytes** – the byte array that needs to be verified

- **path** – the path in which to store the approval file

verify

public static void **verify** (short[] *shorts*, [Path](#) *path*)

An overload for verifying short arrays. This will call the approval object with proper reporter and use the path for verification.

Parameters

- **shorts** – the short array that needs to be verified
- **path** – the path in which to store the approval file

verify

public static void **verify** (long[] *longs*, [Path](#) *path*)

An overload for verifying long arrays. This will call the approval object with proper reporter and use the path for verification.

Parameters

- **longs** – the long array that needs to be verified
- **path** – the path in which to store the approval file

verify

public static void **verify** (float[] *floats*, [Path](#) *path*)

An overload for verifying float arrays. This will call the approval object with proper reporter and use the path for verification.

Parameters

- **floats** – the float array that needs to be verified
- **path** – the path in which to store the approval file

verify

public static void **verify** (double[] *doubles*, [Path](#) *path*)

An overload for verifying double arrays. This will call the approval object with proper reporter and use the path for verification.

Parameters

- **doubles** – the double array that needs to be verified
- **path** – the path in which to store the approval file

verify

public static void **verify** (boolean[] *booleans*, [Path](#) *path*)

An overload for verifying boolean arrays. This will call the approval object with proper reporter and use the path for verification.

Parameters

- **booleans** – the boolean array that needs to be verified
- **path** – the path in which to store the approval file

verify

public static void **verify** (char[] *chars*, *Path path*)

An overload for verifying char arrays. This will call the approval object with proper reporter and use the path for verification.

Parameters

- **chars** – the char array that needs to be verified
- **path** – the path in which to store the approval file

verify

public static void **verify** (String[] *strings*, *Path path*)

An overload for verifying string arrays. This will call the approval object with proper reporter and use the path for verification.

Parameters

- **strings** – the string array that needs to be verified
- **path** – the path in which to store the approval file

verify

public static void **verify** (byte *value*, *Path path*)

An overload for verifying a single byte value. This will call the approval object with proper reporter and use the path for verification.

Parameters

- **value** – the byte that needs to be verified
- **path** – the path in which to store the approval file

verify

public static void **verify** (short *value*, *Path path*)

An overload for verifying a single short value. This will call the approval object with proper reporter and use the path for verification.

Parameters

- **value** – the short that needs to be verified
- **path** – the path in which to store the approval file

verify

public static void **verify** (int *value*, [Path](#) *path*)

An overload for verifying a single int value. This will call the approval object with proper reporter and use the path for verification.

Parameters

- **value** – the int that needs to be verified
- **path** – the path in which to store the approval file

verify

public static void **verify** (long *value*, [Path](#) *path*)

An overload for verifying a single long value. This will call the approval object with proper reporter and use the path for verification.

Parameters

- **value** – the long that needs to be verified
- **path** – the path in which to store the approval file

verify

public static void **verify** (float *value*, [Path](#) *path*)

An overload for verifying a single float value. This will call the approval object with proper reporter and use the path for verification.

Parameters

- **value** – the float that needs to be verified
- **path** – the path in which to store the approval file

verify

public static void **verify** (double *value*, [Path](#) *path*)

An overload for verifying a single double value. This will call the approval object with proper reporter and use the path for verification.

Parameters

- **value** – the double that needs to be verified
- **path** – the path in which to store the approval file

verify

public static void **verify** (boolean *value*, [Path](#) *path*)

An overload for verifying a single boolean value. This will call the approval object with proper reporter and use the path for verification.

Parameters

- **value** – the boolean that needs to be verified

- **path** – the path in which to store the approval file

verify

public static void **verify** (char *value*, Path *path*)

An overload for verifying a single char value. This will call the approval object with proper reporter and use the path for verification.

Parameters

- **value** – the char that needs to be verified
- **path** – the path in which to store the approval file

verify

public static void **verify** (String *value*, Path *path*)

An overload for verifying a single String value. This will call the approval object with proper reporter and use the path for verification.

Parameters

- **value** – the String that needs to be verified
- **path** – the path in which to store the approval file

FullPathMapper

public interface **FullPathMapper**<T>

A mapper that unlike *PathMapper* doesn't resolve the approval file path based on a given sub path but only needs the value. Of course there are possible implementations that don't even need the value like *com.nikolavp.approval.pathmappers.JunitPathMapper*.

Parameters

- **<T>** – the value that will be approved

See also: *PathMapper*

Methods

getApprovalPath

Path **getApprovalPath** (T *value*)

Get the full approval path based on the value.

Parameters

- **value** – the value that will be approved and for which the approval path will be built

Returns a Path for the given value

PathMapper

public interface **PathMapper**<T>

An interface representing objects that will return an appropriate path for the approval process. Most of the times those are used because you don't want to repeat yourself with the same parent path in `com.nikolavp.approval.Approval.verify(Object, java.nio.file.Path)` for the path argument. This will map your approval results file from the value for approval and a possible sub path.

Parameters

- **<T>** – the value that will be approved

See also: `com.nikolavp.approval.pathmappers.ParentPathMapper`

Methods

getPath

`Path` **getPath** (T value, `Path` approvalFilePath)

Gets the path for the approval result based on the value that we want to approve and a sub path for that.

Parameters

- **value** – the value that will be approved
- **approvalFilePath** – a name/subpath for the approval. This will be the path that was passed to `Approval.verify(Object, java.nio.file.Path)`

Returns the full path for the approval result

Pre

public final class **Pre**

Pre conditions exceptions.

Methods

notNull

public static void **notNull** (`Object` value, `String` name)

Verify that a value is not null.

Parameters

- **value** – the value to verify
- **name** – the name of the value that will be used in the exception message.

Reporter

public interface **Reporter**

Created by nikolavp on 1/30/14.

Methods

approveNew

void **approveNew** (byte[] *value*, File *fileForApproval*, File *fileForVerification*)

Called by an `com.nikolavp.approval.Approval` object when a value for verification is produced but no old.

Parameters

- **value** – the new value that came from the verification
- **fileForApproval** – the approval file(this contains the value that was passed in)
- **fileForVerification** – the file for the this new approval value @return true if the new value is approved and false otherwise

canApprove

boolean **canApprove** (File *fileForApproval*)

A method to check if this reporter is supported for the following file type or environment! Reporters are different for different platforms and file types and this in conjunction with `com.nikolavp.approval.reporters.Reporters.firstWorking` will allow you to plug different reporters for different environments(CI, Windows, Linux, MacOS, etc).

Parameters

- **fileForApproval** – the file that we want to approve

Returns true if we can approve the file and false otherwise

notTheSame

void **notTheSame** (byte[] *oldValue*, File *fileForVerification*, byte[] *newValue*, File *fileForApproval*)

Called by an `com.nikolavp.approval.Approval` object when values don't match in the approval process.

Parameters

- **oldValue** – the old value that was found in fileForVerification from old runs
- **newValue** – the new value that was passed for verification
- **fileForVerification** – the file for this approval value
- **fileForApproval** – the file for the new content

com.nikolavp.approval.converters

AbstractConverter

public abstract class **AbstractConverter**<T> implements *Converter*<T>

An abstract class for the Converter interface. All external converters are advised to subclass this class.

Parameters

- **<T>** – the type you want to convert

AbstractStringConverter

public abstract class **AbstractStringConverter**<T> extends *AbstractConverter*<T>

A convenient abstract converter to handle object approvals on string representable objects.

Parameters

- **<T>** – the type you want to convert

Methods

getRawForm

public final byte[] **getRawForm** (T *value*)

getStringForm

protected abstract *String* **getStringForm** (T *value*)

Gets the string representation of the type object. This representation will be written in the files you are going to then use in the approval process.

Parameters

- **value** – the object that you want to convert

Returns the string representation of the object

ArrayConverter

public class **ArrayConverter**<T> extends *AbstractStringConverter*<T[]>

An array converter that uses another converter for it's items. This allows this converter to be composed with another one and allow you to convert your types even if they are in an array. User: nikolavp Date: 20/03/14 Time: 19:34

Parameters

- **<T>** – The type of the items in the list that this converter accepts

Constructors

ArrayConverter

public **ArrayConverter** (*Converter*<T> *typeConverter*)

Creates an array converter that will use the other converter for it's items and just make array structure human readable.

Parameters

- **typeConverter** – the converters for the items in the array

Methods

getStringForm

protected `String` **getStringForm**(`T[] values`)

Converter

public interface **Converter**<`T`>

A converter interface. Converters are the objects in the approval system that convert your object to their raw form that can be written to the files. Note that the raw form is not always a string representation of the object. If for example your object is an image. User: nikolavp Date: 28/02/14 Time: 14:47

Parameters

- **<T>** – the type you are going to convert to raw form

Methods

getRawForm

`byte[]` **getRawForm**(`T value`)

Gets the raw representation of the type object. This representation will be written in the files you are going to then use in the approval process.

Parameters

- **value** – the object that you want to convert

Returns the raw representation of the object

Converters

public final class **Converters**

Converters for primitive types. Most of these just call `toString` on the passed object and then get the raw representation of the string result. . User: nikolavp Date: 28/02/14 Time: 17:25

Fields

BOOLEAN

public static final `Converter`<`Boolean`> **BOOLEAN**

A converter for the primitive or wrapper boolean object.

BOOLEAN_ARRAY

public static final `Converter`<`boolean[]`> **BOOLEAN_ARRAY**

A converter for the primitive boolean arrays.

BYTE

public static final *Converter*<Byte> **BYTE**

A converter for the primitive or wrapper byte types.

BYTE_ARRAY

public static final *Converter*<byte[]> **BYTE_ARRAY**

A converter for the primitive byte arrays.

CHAR

public static final *Converter*<Character> **CHAR**

A converter for the primitive or wrapper char object.

CHAR_ARRAY

public static final *Converter*<char[]> **CHAR_ARRAY**

A converter for the primitive char arrays.

DOUBLE

public static final *Converter*<Double> **DOUBLE**

A converter for the primitive or wrapper double object.

DOUBLE_ARRAY

public static final *Converter*<double[]> **DOUBLE_ARRAY**

A converter for the primitive double arrays.

FLOAT

public static final *Converter*<Float> **FLOAT**

A converter for the primitive or wrapper float object.

FLOAT_ARRAY

public static final *Converter*<float[]> **FLOAT_ARRAY**

A converter for the primitive float arrays.

INTEGER

public static final *Converter*<Integer> **INTEGER**

A converter for the primitive or wrapper int object.

INTEGER_ARRAY

public static final *Converter*<int[]> **INTEGER_ARRAY**
A converter for the primitive int arrays.

LONG

public static final *Converter*<Long> **LONG**
A converter for the primitive or wrapper long object.

LONG_ARRAY

public static final *Converter*<long[]> **LONG_ARRAY**
A converter for the primitive long arrays.

SHORT

public static final *Converter*<Short> **SHORT**
A converter for the primitive or wrapper short object.

SHORT_ARRAY

public static final *Converter*<short[]> **SHORT_ARRAY**
A converter for the primitive short arrays.

STRING

public static final *Converter*<String> **STRING**
A converter for the String object.

STRING_ARRAY

public static final *Converter*<String[]> **STRING_ARRAY**
A converter for an array of strings.

Methods

of

static <T> *Converter*<T> **of** ()

ofArray

static <T> *Converter*<T> **ofArray** ()

DefaultConverter

public class **DefaultConverter** implements *Converter*<byte[]>

Just a simple converter for byte array primitives. We might want to move this into *Converters*. User: nikolavp Date: 28/02/14 Time: 14:54

Methods

getRawForm

public byte[] **getRawForm** (byte[] *value*)

ListConverter

public class **ListConverter**<T> extends *AbstractStringConverter*<List<T>>

A list converter that uses another converter for it's items. This allows this converter to be composed with another one and allow you to convert your types even if they are in a list. User: nikolavp Date: 28/02/14 Time: 17:47

Parameters

- <T> – The type of the items in the list that this converter accepts

Constructors

ListConverter

public **ListConverter** (*Converter*<T> *typeConverter*)

Creates a list converter that will use the other converter for it's items and just make list structure human readable.

Parameters

- **typeConverter** – the converters for the items

Methods

getStringForm

protected String **getStringForm** (List<T> *values*)

ReflectiveBeanConverter

public class **ReflectiveBeanConverter**<T> extends *AbstractStringConverter*<T>

A converter that accepts a bean object and uses reflection to introspect the fields of the bean and builds a raw form of them. Note that the fields must have a human readable string representation for this converter to work properly. User: nikolavp Date: 28/02/14 Time: 15:12

Parameters

- <T> – the type of objects you want convert to it's raw form

Methods

getStringForm

protected `String` **getStringForm** (`T value`)

com.nikolavp.approval.pathmappers

JUnitPathMapper

public class **JUnitPathMapper** implements `TestRule`, `PathMapper`, `FullPathMapper`

A path mapper that have to be declared as a `org.junit.Rule` and will use the standard junit mechanisms to put your approval results in `$package-name-with-slashes/$classname/$methodname`.

This class can be used as a `com.nikolavp.approval.PathMapper` in which case it will put your approval results in that directory or you can use it as a `com.nikolavp.approval.FullPathMapper` in which case your approval result for the **single virifacion** will be put in a file with that path. In the latter case you will have to make sure that there aren't two approvals for a single test method.

Constructors

JUnitPathMapper

public **JUnitPathMapper** (`Path parentPath`)

A parent path in which you want to put your approvals if any.

Parameters

- **parentPath** – the parent path

Methods

apply

public `Statement` **apply** (`Statement base`, `Description description`)

getApprovalPath

public `Path` **getApprovalPath** (`Object value`)

getCurrentTestPath

`Path` **getCurrentTestPath** ()

getPath

public `Path` **getPath** (`Object value`, `Path approvalFilePath`)

ParentPathMapper

public class **ParentPathMapper**<T> implements *PathMapper*<T>

A path mapper that will put all approvals in a common parent path. Let's say you want to put all your approval results in **src/test/resources/approval**(we assume a common maven directory structure) then you can use this mapper as follows:

```
Approval approval = Approval.of(String.class)
    .withPathMapper(new ParentPathMapper(Paths.get("src/test/resources/approval")))
    .build();
```

now the following call

```
approval.verify("Some cool string value", Paths.get("some_cool_value.txt");
```

will put the approved value in the file **src/test/resources/approval/some_cool_value.txt**

Parameters

- **<T>** – the value that will be approved

Constructors

ParentPathMapper

public **ParentPathMapper** (*Path* parentPath)

Creates a parent path mapper that puts approvals in the given parent path.

Parameters

- **parentPath** – the parent path for all approvals

Methods

getPath

public *Path* **getPath** (T value, *Path* approvalFilePath)

com.nikolavp.approval.reporters

AbstractReporter

public abstract class **AbstractReporter** implements *Reporter*

An abstract class for the Reporter interface. All external reporters are advised to subclass this class.

ExecutableDifferenceReporter

public class **ExecutableDifferenceReporter** implements *Reporter*

A reporter that will shell out to an executable that is presented on the user's machine to verify the test output. Note that the approval command and the difference commands can be the same.

- approval command will be used for the first approval

- the difference command will be used when there is already a verified file but it is not the same as the value from the user

Constructors

ExecutableDifferenceReporter

public **ExecutableDifferenceReporter** (*String approvalCommand*, *String diffCommand*)
Main constructor for the executable reporter.

Parameters

- **approvalCommand** – the approval command
- **diffCommand** – the difference command

Methods

approveNew

public void **approveNew** (byte[] *value*, *File approvalDestination*, *File fileForVerification*)

buildApproveNewCommand

protected *String*[] **buildApproveNewCommand** (*File approvalDestination*, *File fileForVerification*)

buildCommandline

static *List*<*String*> **buildCommandline** (*String*... *cmdParts*)

buildNotTheSameCommand

protected *String*[] **buildNotTheSameCommand** (*File fileForVerification*, *File fileForApproval*)

canApprove

public boolean **canApprove** (*File fileForApproval*)

getApprovalCommand

protected *String* **getApprovalCommand** ()

getDiffCommand

protected *String* **getDiffCommand** ()

notTheSame

```
public void notTheSame (byte[] oldValue, File fileForVerification, byte[] newValue, File fileForApproval)
```

runProcess

```
public static Process runProcess (String... cmdParts)
```

Execute a command with the following arguments.

Parameters

- **cmdParts** – the command parts

Throws

- **IOException** – if there were any I/O errors

Returns the process for the command that was started

startProcess

```
Process startProcess (String... cmdParts)
```

FirstWorkingReporter

class **FirstWorkingReporter** implements [Reporter](#)

A reporter that will compose other reporters and use the first one that can approve the objects for verification as per `com.nikolavp.approval.Reporter.canApprove(java.io.File)`.

Constructors

FirstWorkingReporter

```
FirstWorkingReporter (Reporter... others)
```

Methods

approveNew

```
public void approveNew (byte[] value, File fileForApproval, File fileForVerification)
```

canApprove

```
public boolean canApprove (File fileForApproval)
```

notTheSame

```
public void notTheSame (byte[] oldValue, File fileForVerification, byte[] newValue, File fileForApproval)
```

MacOSReporters

public final class **MacOSReporters**

Reporters that use macOS specific binaries, i.e. not cross platform programs.

If you are looking for something cross platform like gvim, emacs, you better look in *com.nikolavp.approval.reporters.Reporters*.

Methods

diffMerge

public static *Reporter* **diffMerge** ()

A reporter that calls *diffmerge* to show you the results.

Returns a reporter that calls diffmerge

ksdiff

public static *Reporter* **ksdiff** ()

A reporter that calls *ksdiff* to show you the results.

Returns a reporter that calls ksdiff

p4merge

public static *Reporter* **p4merge** ()

A reporter that calls *p4merge* to show you the results.

Returns a reporter that calls p4merge

tkdiff

public static *Reporter* **tkdiff** ()

A reporter that calls *tkdiff* to show you the results.

Returns a reporter that calls tkdiff

Reporters

public final class **Reporters**

Created with IntelliJ IDEA. User: nikolavp Date: 10/02/14 Time: 15:10 To change this template use File | Settings | File Templates.

Methods

console

public static *Reporter* **console** ()

Creates a simple reporter that will print/report approvals to the console. This reporter will use convenient

command line tools under the hood to only report the changes in finds. This is perfect for batch modes or when you run your build in a CI server

Returns a reporter that uses console unix tools under the hood

fileLauncher

public static *Reporter* **fileLauncher** ()

A reporter that launches the file under test. This is useful if you for example are generating an spreadsheet and want to verify it.

Returns a reporter that launches the file

firstWorking

public static *Reporter* **firstWorking** (*Reporter*... *others*)

Get a reporter that will use the first working reporter as per `com.nikolavp.approval.Reporter.canApprove` for the reporting.

Parameters

- **others** – an array/list of reporters that will be used

Returns the newly created composite reporter

gedit

public static *Reporter* **gedit** ()

Creates a reporter that uses gedit under the hood for approving.

Returns a reporter that uses gedit under the hood

gvim

public static *Reporter* **gvim** ()

Creates a convenient gvim reporter. This one will use gvimdiff for difference detection and gvim for approving new files. The proper way to exit vim and not approve the new changes is with `":cq"` - just have that in mind!

Returns a reporter that uses vim under the hood

imageMagick

public static *Reporter* **imageMagick** ()

A reporter that compares images. Currently this uses `imagemagick` for comparison. If you only want to view the new image on first approval and when there is a difference, then you better use the `fileLauncher()` reporter which will do this for you.

Returns the reporter that uses ImageMagick for comparison

SwingInteractiveReporter

public class **SwingInteractiveReporter** implements *Reporter*

A reporter that can wrap another reporter and give you a prompt to approve the new result value.

This is useful for reporters that cannot give you a clear way to create the result file. If for example you are using a reporter that only shows you the resulting value but you cannot move it to the proper result file for the approval.

If you say OK/YES on the prompt then the result will be written to the proper file for the next approval time.

Constructors

SwingInteractiveReporter

SwingInteractiveReporter (*Reporter* other, *FileSystemUtils* fileSystemReadWriter)

Methods

approveNew

public void **approveNew** (byte[] value, *File* fileForApproval, *File* fileForVerification)

canApprove

public boolean **canApprove** (*File* fileForApproval)

isHeadless

boolean **isHeadless** ()

notTheSame

public void **notTheSame** (byte[] oldValue, *File* fileForVerification, byte[] newValue, *File* fileForApproval)

promptUser

int **promptUser** ()

wrap

public static *SwingInteractiveReporter* **wrap** (*Reporter* reporter)

Wrap another reporter.

Parameters

- **reporter** – the other reporter

Returns a new reporter that call the other reporter and then prompts the user

WindowsReporters

public final class **WindowsReporters**

Reporters that use windows specific binaries, i.e. the programs that are used are not cross platform.

If you are looking for something cross platform like gvim, emacs, you better look in *com.nikolavp.approval.reporters.Reporters*.

Methods

beyondCompare

public static *Reporter* **beyondCompare** ()

A reporter that calls *Beyond Compare 3* to show you the results.

Returns a reporter that calls beyond compare

notepadPlusPlus

public static *Reporter* **notepadPlusPlus** ()

A reporter that calls *notepad++* to show you the results.

Returns a reporter that calls notepad++

tortoiseImage

public static *Reporter* **tortoiseImage** ()

A reporter that calls *TortoiseIDiff* to show you the results.

Returns a reporter that calls tortoise image difference tool

tortoiseText

public static *Reporter* **tortoiseText** ()

A reporter that calls *TortoiseMerge* to show you the results.

Returns a reporter that calls tortoise difference tool for text

winMerge

public static *Reporter* **winMerge** ()

A reporter that calls *WinMerge* to show you the results.

Returns a reporter that calls win merge

WindowsReporters.WindowsExecutableReporter

public static class **WindowsExecutableReporter** extends *ExecutableDifferenceReporter*

Windows executable reporters should use this class instead of the more general ExecutableDifferenceReporter.

Constructors

WindowsExecutableReporter

public **WindowsExecutableReporter** (*String approvalCommand*, *String diffCommand*)

Main constructor for the executable reporter.

Parameters

- **approvalCommand** – the approval command
- **diffCommand** – the difference command

Methods

canApprove

public boolean **canApprove** (*File fileForApproval*)

com.nikolavp.approval.utils

CrossPlatformCommand

public abstract class **CrossPlatformCommand**<*T*>

A command that when run will execute the proper method for the specified operating system. This is especially useful when you are trying to create for handling different platforms. Here is an example usage of the class:

```
final Boolean result = new CrossPlatformCommand<Boolean>() {
    &#064;Override protected Boolean onWindows() {
        //do your logic for windows
    }

    &#064;Override protected Boolean onUnix() {
        //do your logic for unix
    }

    &#064;Override protected Boolean onMac() {
        //do your logic for mac
    }

    &#064;Override protected Boolean onSolaris() {
        //do your logic for solaris
    }
}.execute();
```

Parameters

- **<T>** – the result from the command.

Methods

execute

public T **execute** ()

Main method that should be executed. This will return the proper result depending on your platform.

Returns the result

isMac

public static boolean **isMac** ()

Check if the current OS is MacOS.

Returns true if macOS and false otherwise

isSolaris

public static boolean **isSolaris** ()

Check if the current OS is Solaris.

Returns true if solaris and false otherwise

isUnix

public static boolean **isUnix** ()

Check if the current OS is some sort of unix.

Returns true if unix and false otherwise

isWindows

public static boolean **isWindows** ()

Check if the current OS is windows.

Returns true if windows and false otherwise

onMac

protected T **onMac** ()

What to execute on macOS.

Returns the result

onSolaris

protected T **onSolaris** ()

What to execute on solaris.

Returns the result

onUnix

protected abstract T **onUnix** ()
What to execute on windows.
Returns the result

onWindows

protected abstract T **onWindows** ()
What to execute on windows.
Returns the result

setOS

static void **setOS** (String *newOs*)

DefaultFileSystemUtils

public class **DefaultFileSystemUtils** implements com.nikolavp.approval.utils.*FileSystemUtils*
A default implementation for *com.nikolavp.approval.utils.FileSystemUtils*. This one just delegates to methods in *Files*. User: nikolavp Date: 27/02/14 Time: 12:26

Methods

createDirectories

public void **createDirectories** (File *directory*)

move

public void **move** (Path *path*, Path *filePath*)

readFully

public byte[] **readFully** (Path *path*)

touch

public void **touch** (Path *pathToCreate*)

write

public void **write** (Path *path*, byte[] *value*)

FileSystemUtils

public interface **FileSystemUtils**

This class is mostly used for indirection in the tests. We just don't like static utility classes. Created by ontotext on 2/2/14.

Methods

createDirectories

void **createDirectories** (*File directory*)

Create a directory and their parent directories as needed.

Parameters

- **directory** – the directory to create

Throws

- **IOException** – if there was an error while creating the directories

move

void **move** (*Path path*, *Path filePath*)

Move a path to another path.

Parameters

- **path** – the source
- **filePath** – the destination

Throws

- **IOException** – if there was an error while moving the paths

readFully

byte[] **readFully** (*Path path*)

Read the specified path as byte array.

Parameters

- **path** – the path to read

Throws

- **IOException** – if there was an error while reading the content

Returns the path content

touch

void **touch** (*Path pathToCreate*)

Creates the specified path with empty content.

Parameters

- **pathToCreate** – the path to create

Throws

- **IOException** – if there was error while creating the path

write

void **write** (*Path* *path*, byte[] *value*)

Write the byte value to the specified path.

Parameters

- **path** – the path
- **value** – the value

Throws

- **IOException** – if there was an error while writing the content

A

AbstractConverter (Java class), 20
AbstractReporter (Java class), 27
AbstractStringConverter (Java class), 21
apply(Statement, Description) (Java method), 26
Approval (Java class), 11
Approval(Reporter, Converter, PathMapper) (Java constructor), 12
Approval(Reporter, Converter, PathMapper, com.nikolavp.approval.utils.FileSystemUtils) (Java constructor), 12
ApprovalBuilder (Java class), 13
Approvals (Java class), 14
approveNew(byte[], File, File) (Java method), 20, 28, 29, 32
ArrayConverter (Java class), 21
ArrayConverter(Converter) (Java constructor), 21

B

beyondCompare() (Java method), 33
BOOLEAN (Java field), 22
BOOLEAN_ARRAY (Java field), 22
build() (Java method), 13
buildApproveNewCommand(File, File) (Java method), 28
buildCommandline(String) (Java method), 28
buildNotTheSameCommand(File, File) (Java method), 28
BYTE (Java field), 23
BYTE_ARRAY (Java field), 23

C

canApprove(File) (Java method), 20, 28, 29, 32, 34
CHAR (Java field), 23
CHAR_ARRAY (Java field), 23
com.nikolavp.approval (package), 11
com.nikolavp.approval.converters (package), 20
com.nikolavp.approval.pathmappers (package), 26
com.nikolavp.approval.reporters (package), 27
com.nikolavp.approval.utils (package), 34

console() (Java method), 30
Converter (Java interface), 22
Converters (Java class), 22
createDirectories(File) (Java method), 36, 37
CrossPlatformCommand (Java class), 34

D

DefaultConverter (Java class), 25
DefaultFileSystemUtils (Java class), 36
diffMerge() (Java method), 30
DOUBLE (Java field), 23
DOUBLE_ARRAY (Java field), 23

E

ExecutableDifferenceReporter (Java class), 27
ExecutableDifferenceReporter(String, String) (Java constructor), 28
execute() (Java method), 35

F

fileLauncher() (Java method), 31
FileSystemUtils (Java interface), 37
firstWorking(Reporter) (Java method), 31
FirstWorkingReporter (Java class), 29
FirstWorkingReporter(Reporter) (Java constructor), 29
FLOAT (Java field), 23
FLOAT_ARRAY (Java field), 23
FullPathMapper (Java interface), 18

G

gedit() (Java method), 31
getApprovalCommand() (Java method), 28
getApprovalPath(Object) (Java method), 26
getApprovalPath(Path) (Java method), 12
getApprovalPath(T) (Java method), 18
getConverter() (Java method), 12
getCurrentTestPath() (Java method), 26
getDiffCommand() (Java method), 28
getPath(Object, Path) (Java method), 26

getPath(T, Path) (Java method), 19, 27
getPathMapper() (Java method), 12
getRawForm(byte[]) (Java method), 25
getRawForm(T) (Java method), 21, 22
getReporter() (Java method), 12
getStringForm(List) (Java method), 25
getStringForm(T) (Java method), 21, 26
getStringForm(T[]) (Java method), 22
gvim() (Java method), 31

I

imageMagick() (Java method), 31
INTEGER (Java field), 23
INTEGER_ARRAY (Java field), 24
isHeadless() (Java method), 32
isMac() (Java method), 35
isSolaris() (Java method), 35
isUnix() (Java method), 35
isWindows() (Java method), 35

J

JUnitPathMapper (Java class), 26
JUnitPathMapper(Path) (Java constructor), 26

K

ksdiff() (Java method), 30

L

ListConverter (Java class), 25
ListConverter(Converter) (Java constructor), 25
LONG (Java field), 24
LONG_ARRAY (Java field), 24

M

MacOSReporters (Java class), 30
move(Path, Path) (Java method), 36, 37

N

notepadPlusPlus() (Java method), 33
notNull(Object, String) (Java method), 19
notTheSame(byte[], File, byte[], File) (Java method), 20,
29, 32

O

of() (Java method), 24
of(Class) (Java method), 13
ofArray() (Java method), 24
onMac() (Java method), 35
onSolaris() (Java method), 35
onUnix() (Java method), 36
onWindows() (Java method), 36

P

p4merge() (Java method), 30

ParentPathMapper (Java class), 27
ParentPathMapper(Path) (Java constructor), 27
PathMapper (Java interface), 19
Pre (Java class), 19
promptUser() (Java method), 32

R

readFully(Path) (Java method), 36, 37
ReflectiveBeanConverter (Java class), 25
Reporter (Java interface), 19
Reporters (Java class), 30
runProcess(String) (Java method), 29

S

setOS(String) (Java method), 36
SHORT (Java field), 24
SHORT_ARRAY (Java field), 24
startProcess(String) (Java method), 29
STRING (Java field), 24
STRING_ARRAY (Java field), 24
SwingInteractiveReporter (Java class), 32
SwingInteractiveReporter(Reporter, FileSystemUtils)
(Java constructor), 32

T

tkdiff() (Java method), 30
tortoiseImage() (Java method), 33
tortoiseText() (Java method), 33
touch(Path) (Java method), 36, 37

V

verify(boolean, Path) (Java method), 17
verify(boolean[], Path) (Java method), 15
verify(byte, Path) (Java method), 16
verify(byte[], Path) (Java method), 14
verify(char, Path) (Java method), 18
verify(char[], Path) (Java method), 16
verify(double, Path) (Java method), 17
verify(double[], Path) (Java method), 15
verify(float, Path) (Java method), 17
verify(float[], Path) (Java method), 15
verify(int, Path) (Java method), 17
verify(int[], Path) (Java method), 14
verify(long, Path) (Java method), 17
verify(long[], Path) (Java method), 15
verify(short, Path) (Java method), 16
verify(short[], Path) (Java method), 15
verify(String, Path) (Java method), 18
verify(String[], Path) (Java method), 16
verify(T, Path) (Java method), 13

W

WindowsExecutableReporter (Java class), 33

WindowsExecutableReporter(String, String) (Java constructor), [34](#)
WindowsReporters (Java class), [33](#)
winMerge() (Java method), [33](#)
withConveter(Converter) (Java method), [13](#)
withPathMapper(PathMapper) (Java method), [14](#)
withReporter(Reporter) (Java method), [14](#)
wrap(Reporter) (Java method), [32](#)
write(Path, byte[]) (Java method), [36](#), [38](#)