See the Assessment Guide for information on how to interpret this report.

ASSESSMENT SUMMARY

```
Compilation:  PASSED
API:          PASSED

SpotBugs:     FAILED (1 warning)
PMD:          PASSED
Checkstyle:   FAILED (0 errors, 1 warning)

Correctness:  40/40 tests passed
Memory:       No tests available for autograding.
Timing:       No tests available for autograding.

Aggregate score: 100.00%
[ Compilation: 5%, API: 5%, Style: 0%, Correctness: 90% ]
```

ASSESSMENT DETAILS

```
The following files were submitted:
----------------------------------
1.9K Jul  4 20:04 Huntingtons.java
6.0K Jul  4 20:04 KernelFilter.java


================================================================================
*  COMPILING
================================================================================


% javac Huntingtons.java
*------------------------------------------------------------

% javac KernelFilter.java
*------------------------------------------------------------


================================================================


Checking the APIs of your programs.
*------------------------------------------------------------
Huntingtons:

KernelFilter:

================================================================


================================================================================
*  CHECKING STYLE AND COMMON BUG PATTERNS
================================================================================


% spotbugs *.class
*------------------------------------------------------------
M C SUA_SUSPICIOUS_UNINITIALIZED_ARRAY SUA: Method KernelFilter.convertPicturetoMatrix(Picture) returns an array that appears not to be initi
SpotBugs ends with 1 warning.


================================================================


% pmd .
*------------------------------------------------------------


================================================================


% checkstyle *.java
*------------------------------------------------------------

% custom checkstyle checks for Huntingtons.java
*------------------------------------------------------------

% custom checkstyle checks for KernelFilter.java
*------------------------------------------------------------
[WARN] KernelFilter.java:38:51: '0.5' looks like an unnecessary constant. [MagicNumber]
Checkstyle ends with 0 errors and 1 warning.


================================================================


================================================================================
```

```
*  TESTING CORRECTNESS
*********************************************************************************

Testing correctness of Huntingtons
*-----------------------------------------------------------
Running 10 total tests.

Test 1: check output format of main() for inputs from assignment specification
  % java-introcs Huntingtons repeats4.txt
  max repeats = 4
  not human

  % java-introcs Huntingtons repeats64.txt
  max repeats = 64
  Huntington's

  % java-introcs Huntingtons chromosome4-hd.txt
  max repeats = 79
  Huntington's

  % java-introcs Huntingtons chromosome4-healthy.txt
  max repeats = 19
  normal

==> passed

Test 2: check correctness of main() for inputs from assignment specification
  % java-introcs Huntingtons repeats4.txt
  % java-introcs Huntingtons repeats64.txt
  % java-introcs Huntingtons chromosome4-hd.txt
  % java-introcs Huntingtons chromosome4-healthy.txt
==> passed

Test 3: check maxRepeats() for DNA from files (with whitespace removed)
  * file = repeats0.txt
  * file = repeats2.txt
  * file = repeats4.txt
  * file = repeats9.txt
  * file = repeats10.txt
  * file = repeats12.txt
  * file = repeats35.txt
  * file = repeats36.txt
  * file = repeats39.txt
  * file = repeats40.txt
  * file = repeats64.txt
  * file = repeats180.txt
  * file = repeats181.txt
==> passed

Test 4: check maxRepeats() for DNA from files (with whitespace removed)
  * file = chromosome4-hd.txt
  * file = chromosome4-healthy.txt
==> passed

Test 5: check maxRepeats() for random DNA of length n
  * 10000 random strings of length 10
  * 10000 random strings of length 20
  * 10000 random strings of length 30
  * 10000 random strings of length 100
  * 10000 random strings of length 200
  * 10000 random strings of length 500
==> passed

Test 6: check removeWhitespace() for inputs from files
  * file = repeats0.txt
  * file = repeats2.txt
  * file = repeats4.txt
  * file = repeats9.txt
  * file = repeats10.txt
  * file = repeats12.txt
  * file = repeats35.txt
  * file = repeats36.txt
  * file = repeats39.txt
  * file = repeats40.txt
  * file = repeats64.txt
  * file = repeats180.txt
  * file = repeats181.txt
==> passed

Test 7: check removeWhitespace() for DNA from files
  * file = chromosome4-hd.txt
  * file = chromosome4-healthy.txt
==> passed

Test 8: check maxRepeats() for random DNA of length n
  * 10000 random strings of length 10 over alphabet { 'A', 'C', 'G', 'T' }
  * 10000 random strings of length 10 over alphabet { 'A', 'C', 'G', 'T', ' ' }
  * 10000 random strings of length 10 over alphabet { 'A', 'C', 'G', 'T', ' ', '\n' }
  * 10000 random strings of length 10 over alphabet { 'A', 'C', 'G', 'T', ' ', '\n', '\t' }
  * 10000 random strings of length 20 over alphabet { 'A', 'C', 'G', 'T' }
  * 10000 random strings of length 20 over alphabet { 'A', 'C', 'G', 'T', ' ' }
  * 10000 random strings of length 20 over alphabet { 'A', 'C', 'G', 'T', ' ', '\n' }
  * 10000 random strings of length 20 over alphabet { 'A', 'C', 'G', 'T', ' ', '\n', '\t' }
  * 10000 random strings of length 100 over alphabet { 'A', 'C', 'G', 'T' }
  * 10000 random strings of length 100 over alphabet { 'A', 'C', 'G', 'T', ' ' }
  * 10000 random strings of length 100 over alphabet { 'A', 'C', 'G', 'T', ' ', '\n' }
  * 10000 random strings of length 100 over alphabet { 'A', 'C', 'G', 'T', ' ', '\n', '\t' }
==> passed
```

```
Test 9: check diagnose() for given value of maxRepeats
  * maxRepeats = 0
  * maxRepeats = 9
  * maxRepeats = 10
  * maxRepeats = 35
  * maxRepeats = 36
  * maxRepeats = 39
  * maxRepeats = 40
  * maxRepeats = 180
  * maxRepeats = 181
==> passed

Test 10: check diagnose() for range of values of maxRepeats
  * 0 to 9
  * 10 to 35
  * 36 to 39
  * 40 to 180
  * 180 to 1000
==> passed


Huntingtons Total: 10/10 tests passed!


*-----------------------------------------------------------
Testing correctness of KernelFilter
*-----------------------------------------------------------
Running 30 total tests.

Test 1: check correctness of identity() for given grayscale PNG files
  * 6-by-5.png
  * baboon-gray.png
  * sunflowers-gray.png
  * earth-gray.png
  * penguins-gray.png
==> passed

Test 2: check correctness of identity() for given color PNG files
  * baboon.png
  * baboon-red.png
  * baboon-green.png
  * baboon-blue.png
  * sunflowers.png
  * earth.png
  * penguins.png
==> passed

Test 3: check correctness of identity() for random grayscale pictures
  * 1000 random 9-by-9 grayscale images
  * 1000 random 5-by-8 grayscale images
  * 1000 random 7-by-6 grayscale images
  * 1000 random 1-by-8 grayscale images
  * 1000 random 8-by-1 grayscale images
  * 1000 random 1-by-1 grayscale images
==> passed

Test 4: check correctness of identity() for random color pictures
  * 1000 random 10-by-10 color images
  * 1000 random 12-by-17 color images
  * 1000 random 16-by-13 color images
==> passed

Test 5: check correctness of gaussian() for given grayscale PNG files
  * 6-by-5.png
  * baboon-gray.png
  * sunflowers-gray.png
  * earth-gray.png
  * penguins-gray.png
==> passed

Test 6: check correctness of gaussian() for given color PNG files
  * baboon.png
  * baboon-red.png
  * baboon-green.png
  * baboon-blue.png
  * sunflowers.png
  * earth.png
  * penguins.png
==> passed

Test 7: check correctness of gaussian() for random grayscale pictures
  * 1000 random 9-by-9 grayscale images
  * 1000 random 5-by-8 grayscale images
  * 1000 random 7-by-6 grayscale images
  * 1000 random 1-by-8 grayscale images
  * 1000 random 8-by-1 grayscale images
  * 1000 random 1-by-1 grayscale images
==> passed

Test 8: check correctness of gaussian() for random color pictures
  * 1000 random 10-by-10 color images
  * 1000 random 12-by-17 color images
  * 1000 random 16-by-13 color images
==> passed

Test 9: check correctness of sharpen() for given grayscale PNG files
  * 6-by-5.png
```

```
    * baboon-gray.png
    * sunflowers-gray.png
    * earth-gray.png
    * penguins-gray.png
  ==> passed

  Test 10: check correctness of sharpen() for given color PNG files
    * baboon.png
    * baboon-red.png
    * baboon-green.png
    * baboon-blue.png
    * sunflowers.png
    * earth.png
    * penguins.png
  ==> passed

  Test 11: check correctness of sharpen() for random grayscale pictures
    * 1000 random 9-by-9 grayscale images
    * 1000 random 5-by-8 grayscale images
    * 1000 random 7-by-6 grayscale images
    * 1000 random 1-by-8 grayscale images
    * 1000 random 8-by-1 grayscale images
    * 1000 random 1-by-1 grayscale images
  ==> passed

  Test 12: check correctness of sharpen() for random color pictures
    * 1000 random 10-by-10 color images
    * 1000 random 12-by-17 color images
    * 1000 random 16-by-13 color images
  ==> passed

  Test 13: check correctness of laplacian() for given grayscale PNG files
    * 6-by-5.png
    * baboon-gray.png
    * sunflowers-gray.png
    * earth-gray.png
    * penguins-gray.png
  ==> passed

  Test 14: check correctness of laplacian() for given color PNG files
    * baboon.png
    * baboon-red.png
    * baboon-green.png
    * baboon-blue.png
    * sunflowers.png
    * earth.png
    * penguins.png
  ==> passed

  Test 15: check correctness of laplacian() for random grayscale pictures
    * 1000 random 9-by-9 grayscale images
    * 1000 random 5-by-8 grayscale images
    * 1000 random 7-by-6 grayscale images
    * 1000 random 1-by-8 grayscale images
    * 1000 random 8-by-1 grayscale images
    * 1000 random 1-by-1 grayscale images
  ==> passed

  Test 16: check correctness of laplacian() for random color pictures
    * 1000 random 10-by-10 color images
    * 1000 random 12-by-17 color images
    * 1000 random 16-by-13 color images
  ==> passed

  Test 17: check correctness of emboss() for given grayscale PNG files
    * 6-by-5.png
    * baboon-gray.png
    * sunflowers-gray.png
    * earth-gray.png
    * penguins-gray.png
  ==> passed

  Test 18: check correctness of emboss() for given color PNG files
    * baboon.png
    * baboon-red.png
    * baboon-green.png
    * baboon-blue.png
    * sunflowers.png
    * earth.png
    * penguins.png
  ==> passed

  Test 19: check correctness of emboss() for random grayscale pictures
    * 1000 random 9-by-9 grayscale images
    * 1000 random 5-by-8 grayscale images
    * 1000 random 7-by-6 grayscale images
    * 1000 random 1-by-8 grayscale images
    * 1000 random 8-by-1 grayscale images
    * 1000 random 1-by-1 grayscale images
  ==> passed

  Test 20: check correctness of emboss() for random color pictures
    * 1000 random 10-by-10 color images
    * 1000 random 12-by-17 color images
    * 1000 random 16-by-13 color images
  ==> passed

  Test 21: check correctness of motionBlur() for given grayscale PNG files
```

```
    * 6-by-5.png
    * baboon-gray.png
    * sunflowers-gray.png
    * earth-gray.png
    * penguins-gray.png
  ==> passed

  Test 22: check correctness of motionBlur() for given color PNG files
    * baboon.png
    * baboon-red.png
    * baboon-green.png
    * baboon-blue.png
    * sunflowers.png
    * earth.png
    * penguins.png
  ==> passed

  Test 23: check correctness of motionBlur() for random grayscale pictures
    * 1000 random 9-by-9 grayscale images
    * 1000 random 5-by-8 grayscale images
    * 1000 random 7-by-6 grayscale images
    * 1000 random 1-by-8 grayscale images
    * 1000 random 8-by-1 grayscale images
    * 1000 random 1-by-1 grayscale images
  ==> passed

  Test 24: check correctness of motionBlur() for random color pictures
    * 1000 random 10-by-10 color images
    * 1000 random 12-by-17 color images
    * 1000 random 16-by-13 color images
  ==> passed

  Test 25: check that identity() does not mutate Picture argument
    * baboon.png
    * sunflowers.png
    * earth.png
    * penguins.png
  ==> passed

  Test 26: check that gaussian() does not mutate Picture argument
    * baboon.png
    * sunflowers.png
    * earth.png
    * penguins.png
  ==> passed

  Test 27: check that sharpen() does not mutate Picture argument
    * baboon.png
    * sunflowers.png
    * earth.png
    * penguins.png
  ==> passed

  Test 28: check that laplacian() does not mutate Picture argument
    * baboon.png
    * sunflowers.png
    * earth.png
    * penguins.png
  ==> passed

  Test 29: check that emboss() does not mutate Picture argument
    * baboon.png
    * sunflowers.png
    * earth.png
    * penguins.png
  ==> passed

  Test 30: check that motionBlur() does not mutate Picture argument
    * baboon.png
    * sunflowers.png
    * earth.png
    * penguins.png
  ==> passed


  KernelFilter Total: 30/30 tests passed!


  ================================================================
```