Ein weiteres Einreichungsbeispiel zum "JUnit-Backend" - Primes.java -

Christian Baumann

12. Januar 2008

Inhaltsverzeichnis

1 Fehlerlose Beispieleinreichung:

```java
/**
 * This class is used to compute prime numbers.
 * @author christian
 *
 */
public class Prime {
    /**
     * Counts prime numbers up to a specified bound.
     * @param bound Count up to this number.
     * @return Number of primes.
     */
    public int countPrimesUpTo(int bound){
        int count = 0;

        for(int i = 2; i <= bound; i++){
            if (isPrime(i)) count++;
        }

        return count;
    }

    /**
     * Checks whether an integer is a prime number or not.
     * @param number Number to be checked.
     * @return true if number is prime, false otherwise.
     */
    public boolean isPrime(int number){
        if(number == 1){
            return false;
        }else if(number == 2){
            return true;
        }else if(number % 2 == 0){
            return false;
        }else{
            for(int i = 2; i < Math.sqrt(number)+1; i += 1){
                if (number % i == 0) return false;
            }
        }
        return true;
    }

    /**
     * Returns a formatted string of the first prime numbers up to bound.
     * (e.g. '[2, 3, 5, 7]')
     * @param bound Highest number that should be tested.
     * @return String containing prime numbers up to bound.
     */
    public String getStringOfPrimesUpTo(int bound){
        boolean format = false;
        String string = "[";

        for(int i = 2; i <= bound; i++){
            if (this.isPrime(i)){
                if (format) string += ", ";
                string += i;
                format = true;
            }
        }

        return string += "]";
    }
}
```

2 Einreichung mit syntaktischem Fehler:

```
1  /**
2   * This class is used to compute prime numbers and has a
3   * syntactical error in line 14.
4   * @author christian
5   *
6   */
7  public class Prime {
8      /**
9       * Counts prime numbers up to a specified bound.
10      * @param bound Count up to this number.
11      * @return Number of primes.
12      */
13     public int countPrimesUpTo(int bound){
14         int count = 0
15
16         for(int i = 2; i <= bound; i++){
17             if (isPrime(i)) count++;
18         }
19
20         return count;
21     }
22
23     /**
24      * Checks whether an integer is a prime number or not.
25      * @param number Number to be checked.
26      * @return true if number is prime, false otherwise.
27      */
28     public boolean isPrime(int number){
29         if(number == 1){
30             return false;
31         }else if(number == 2){
32             return true;
33         }else if(number % 2 == 0){
34             return false;
35         }else{
36             for(int i = 2; i < Math.sqrt(number)+1; i += 1){
37                 if (number % i == 0) return false;
38             }
39         }
40         return true;
41     }
42
43     /**
44      * Returns a formatted string of the first prime numbers up to bound.
45      * (e.g. '[2, 3, 5, 7]')
46      * @param bound Highest number that should be tested.
47      * @return String containing prime numbers up to bound.
48      */
49     public String getStringOfPrimesUpTo(int bound){
50         boolean format = false;
51         String string = "[";
52
53         for(int i = 2; i <= bound; i++){
54             if (this.isPrime(i)){
55                 if (format) string += ", ";
56                 string += i;
57                 format = true;
58             }
59         }
60
61         return string += "]";
62     }
63 }
```

3 Einreichung mit semantischem Fehler:

```
 1  /**
 2   * This class is used to compute prime numbers and has an semantical error in
 3   * line 16, since primes are only tested for values below 'bound'.
 4   * @author christian
 5   *
 6   */
 7  public class Prime {
 8      /**
 9       * Counts prime numbers up to a specified bound.
10       * @param bound Count up to this number.
11       * @return Number of primes.
12       */
13      public int countPrimesUpTo(int bound){
14          int count = 0;
15
16          for(int i = 2; i < bound; i++){
17              if (isPrime(i)) count++;
18          }
19
20          return count;
21      }
22
23      /**
24       * Checks whether an integer is a prime number or not.
25       * @param number Number to be checked.
26       * @return true if number is prime, false otherwise.
27       */
28      public boolean isPrime(int number){
29          if(number == 1){
30              return false;
31          }else if(number == 2){
32              return true;
33          }else if(number % 2 == 0){
34              return false;
35          }else{
36              for(int i = 2; i < Math.sqrt(number)+1; i += 1){
37                  if (number % i == 0) return false;
38              }
39          }
40          return true;
41      }
42
43      /**
44       * Returns a formatted string of the first prime numbers up to bound.
45       * (e.g. '[2, 3, 5, 7]')
46       * @param bound Highest number that should be tested.
47       * @return String containing prime numbers up to bound.
48       */
49      public String getStringOfPrimesUpTo(int bound){
50          boolean format = false;
51          String string = "[";
52
53          for(int i = 2; i <= bound; i++){
54              if (this.isPrime(i)){
55                  if (format) string += ",␣";
56                  string += i;
57                  format = true;
58              }
59          }
60
61          return string += "]";
62      }
63  }
```

4 Unit-Tests:

```
@Test public void testHowManyPrimes(){
    ${CLASS} c = new ${CLASS}();
    assertEquals("The number of counted primes is wrong.",
                 c.countPrimesUpTo(541), 100);
}

@Test public void testSeveralKnownPrimes(){
    int[] primes = {2,3,5,7,541,191,193,197,199};

    ${CLASS} c = new ${CLASS}();

    for(int i = 0; i < primes.length; i++){
        assertTrue(c.isPrime(primes[i]));
    }
}

@Test public void testSeveralNumbersKnownAsNonPrimes(){
    int[] nonprimes = {1,4,6,9,99,540,1024};

    ${CLASS} c = new ${CLASS}();

    for(int i = 0; i < nonprimes.length; i++){
        assertFalse(c.isPrime(nonprimes[i]));
    }
}

@Test public void testReturnedString(){
    String string = "";
    ${CLASS} c = new ${CLASS}();

    //Does it return a string?
    try{
        string = c.getStringOfPrimesUpTo(541);
    }catch (Exception e){
        fail("Method 'getStringOfPrimesUpTo' does not return a String");
    }

    //Does it have the desired format?
    //e.g. '[]' or '[2, 3, 5, 7]'
    //Testing if the string starts with [ and ends with ]:
    assertEquals("The returned string of primes does not start with '['.",
                 string.charAt(0), '[');
    assertEquals("The returned string of primes does not end with ']'.",
                 string.charAt(string.length()-1), ']');
}
```