Some Important Heuristic Tests for HW4

Test the Constructor for Off-By-One Faults

```
try {
    // z is the last letter of the alphabet and is missing
    new PhraseCypher("the quick brown fox jumps over the lady dog");
    fail("An IllegalArgumentException should have been thrown (but wasn't).");
} catch (IllegalArgumentException iae) {
    // This is what should happen!
    String message = iae.getMessage();
    char expected = 'Z';
    char actual = message.charAt(message.length()-1);
    assertEquals(expected, actual);
}
```

```
try {
    // space is the first letter of the alphabet and is missing
    new PhraseCypher("thequickbrownfoxjumpsovertheladydog");
    fail("An IllegalArgumentException should have been thrown (but wasn't).");
} catch (IllegalArgumentException iae) {
    // This is what should happen!
    String message = iae.getMessage();
    char expected = ' ';
    char actual = message.charAt(message.length()-1);
    assertEquals(expected, actual);
}
```

Test the Constructor for Case Independence

```
try {
    new PhraseCypher("THE QUICK BROWN FOX JUMPS OVER THE LADY DOG");
    fail("An IllegalArgumentException should have been thrown (but wasn't).");
} catch (IllegalArgumentException iae) {
    // This is what should happen!
    String message = iae.getMessage();
    char expected = 'Z';
    char actual = message.charAt(message.length()-1);
    assertEquals(expected, actual);
}
```

```
try {
    new PhraseCypher("THEQUICKBROWNFOXJUMPSOVERTHELADYDOG");
    fail("An IllegalArgumentException should have been thrown (but wasn't).");
} catch (IllegalArgumentException iae) {
    // This is what should happen!
    String message = iae.getMessage();
    char expected = ' ';
    char actual = message.charAt(message.length()-1);
    assertEquals(expected, actual);
}
```

Test the Constructor for a `null key`

```
try {
    new PhraseCypher(null);
    fail("An IllegalArgumentException should have been thrown (but wasn't).");
} catch (IllegalArgumentException iae) {
    // This is what should happen!
    String message = iae.getMessage();
    char expected = ' ';
    char actual = message.charAt(message.length()-1);
    assertEquals(expected, actual);
}
```

Test the Constructor for an Invalid `key`

```
try {
    new PhraseCypher("");
    fail("An IllegalArgumentException should have been thrown (but wasn't).");
} catch (IllegalArgumentException iae) {
    // This is what should happen!
    String message = iae.getMessage();
    char expected = ' ';
    char actual = message.charAt(message.length()-1);
    assertEquals(expected, actual);
}

try {
    new PhraseCypher("THE QUICK BROWN FOX JUMPS 1 TIME OVER THE LAZY DOG");
    fail("An IllegalArgumentException should have been thrown (but wasn't).");
} catch (IllegalArgumentException iae) {
    // This is what should happen!
    String message = iae.getMessage();
    char expected = '1';
    char actual = message.charAt(message.length()-1);
    assertEquals(expected, actual);
}
```

Test the `encrypt()` Method

```
PhraseCypher cypher;
//                    1         2         3         4
//          01234567890123456789012345678901234567890123456789012
String s = "the quick brown fox jumps over the lazy dog";
cypher = new PhraseCypher(s);

{
    int[] actual = cypher.encrypt(null);
    int[] expected = new int[0];

    assertArrayEquals(expected, actual);
}
{
    int[] actual = cypher.encrypt("");
    int[] expected = new int[0];

    assertArrayEquals(expected, actual);
}
```

```
{
    int[] actual = cypher.encrypt("FOR NOW");
    //                 F   O   R      N   O   W
    int[] expected = {16, 12, 11, 3, 14, 12, 13};

    assertArrayEquals(expected, actual);
}
{
    int[] actual = cypher.encrypt("for now");
    int[] expected = {16, 12, 11, 3, 14, 12, 13};

    assertArrayEquals(expected, actual);
}
{
    int[] actual = cypher.encrypt("FoR nOw");
    int[] expected = {16, 12, 11, 3, 14, 12, 13};

    assertArrayEquals(expected, actual);
}
{
    int[] actual = cypher.encrypt("fOr NoW");
    int[] expected = {16, 12, 11, 3, 14, 12, 13};

    assertArrayEquals(expected, actual);
}

//             1         2
//   012345678901234567890123456
s = " ABCDEFGHIJKLMNOPQRSTUVWXYZ";
cypher = new PhraseCypher(s);
{
    int[] actual = cypher.encrypt(s);
    int[] expected = new int[27];
    for (int i=0; i<expected.length; i++) expected[i] = i;

    assertArrayEquals(expected, actual);
}
{
    int[] actual = cypher.encrypt("ZYXWVUTSRQPONMLKJIHGFEDCBA ");
    int[] expected = new int[27];
    for (int i=0; i<expected.length; i++) expected[i] = 26 - i;

    assertArrayEquals(expected, actual);
}
{
    int[] actual = cypher.encrypt("     ");
    int[] expected = new int[]{0, 0, 0, 0, 0};

    assertArrayEquals(expected, actual);
}
{
    int[] actual = cypher.encrypt("ZZZ");
    int[] expected = new int[]{26, 26, 26};

    assertArrayEquals(expected, actual);
}
{
    int[] actual = cypher.encrypt("A1B");
    int[] expected = new int[]{1, -1, 2};

    assertArrayEquals(expected, actual);
}
```

Test the `decrypt()` Method

```
PhraseCypher cypher;
//                    1         2         3         4
//          01234567890123456789012345678901234567890123
String s = "the quick brown fox jumps over the lazy dog";
cypher = new PhraseCypher(s);

{
    String actual = cypher.decrypt(new int[0]);
    String expected = "";

    assertEquals(expected, actual);
}
{
    String actual = cypher.decrypt(null);
    String expected = "";

    assertEquals(expected, actual);
}
{
    String actual = cypher.decrypt(new int[]{0, 0, 0, 0, 0, 0, 0, 0, 0, 0});
    String expected = "TTTTTTTTTT";

    assertEquals(expected, actual);
}
{
    String actual = cypher.decrypt(new int[]{26, 26, 26, 26});
    String expected = "OOOO";

    assertEquals(expected, actual);
}
```