```
00001:
00002: import java.io.ByteArrayOutputStream;
00003: import java.io.File;
00004: import java.io.PrintStream;
00005: import java.util.Scanner;
00006: import org.junit.Test;
00007: import static org.junit.Assert.*;
00008:
00009: /**
00010:  * This Test is for Cipher.java and it test inputs for the file
00011:  *
00012:  * @author WahabEhsan
00013:  */
00014:
00015: public class CipherTest {
00016:
00017:     /**
00018:      * Variable for new line.
00019:      */
00020:     public static final String NEWLINE = System.getProperty("line.separator");
00021:
00022:     private static final String taleOriginal = "It was the best of times," + NEWLINE
00023:             + "it was the worst of times," + NEWLINE
00024:             + "it was the age of wisdom," + NEWLINE
00025:             + "it was the age of foolishness," + NEWLINE
00026:             + "it was the epoch of belief," + NEWLINE
00027:             + "it was the epoch of incredulity," + NEWLINE
00028:             + "it was the season of Light," + NEWLINE
00029:             + "it was the season of Darkness," + NEWLINE
00030:             + "it was the spring of hope," + NEWLINE
00031:             + "it was the winter of despair, ...";
00032:     private static final String taleEncrypted = "Nn jfm gmy ojmg tz gngrx," + NEWLINE
00033:             + "nn jfm gmy jtlfy is yczjm," + NEWLINE
```

```
00034:             + "nn jfm gmy nly bk qvxxbr," + NEWLINE
00035:             + "nn jfm gmy nly bk zbtfvxbajmf," + NEWLINE
00036:             + "nn jfm gmy ruipm is gyynys," + NEWLINE
00037:             + "nn jfm gmy ruipm is nhpwyqzfvys," + NEWLINE
00038:             + "nn jfm gmy fjufth bk Fvlbg," + NEWLINE
00039:             + "nn jfm gmy fjufth bk Xnweajmf," + NEWLINE
00040:             + "nn jfm gmy fulvsa bk bbuy," + NEWLINE
00041:             + "nn jfm gmy jnhgjl bk xrxjnnl, ...";
00042: 
00043:     /**
00044:      * Checks getKey for valid key input.
00045:      */
00046:     @Test
00047:     public void testGetKeyWithValidKey() {
00048:         Scanner console = new Scanner("validkey");
00049:         String expResult = "validkey";
00050:         String result = Cipher.getKey(console);
00051:         assertEquals(expResult, result);
00052:     }
00053: 
00054:     /**
00055:      * Checks getKey for invalid key input.
00056:      */
00057:     @Test
00058:     public void testGetKeyWithInvalidKeyFirst() {
00059:         Scanner console = new Scanner("invalidKey" + NEWLINE + "validkey");
00060:         String expResult = "validkey";
00061:         String result = Cipher.getKey(console);
00062:         assertEquals(expResult, result);
00063:     }
00064: 
00065:     /**
00066:      * Checks getKey for invalid key as number input.
```

```
00067:      */
00068:     @Test
00069:     public void testGetKeyWithNumericKeysFirst() {
00070:         Scanner console = new Scanner("2" + NEWLINE + "validkey");
00071:         String expResult = "validkey";
00072:         String result = Cipher.getKey(console);
00073:         assertEquals(expResult, result);
00074:     }
00075:
00076:     /**
00077:      * Checks getKey for invalid key as empty input.
00078:      */
00079:     @Test
00080:     public void testGetKeyWithEmptyKeysFirst() {
00081:         Scanner console = new Scanner("" + NEWLINE + "validkey");
00082:         String expResult = "validkey";
00083:         String result = Cipher.getKey(console);
00084:         assertEquals(expResult, result);
00085:     }
00086:
00087:     /**
00088:      * Checks getKey for invalid key as spaces in input.
00089:      */
00090:     @Test
00091:     public void testGetKeyWithKeyWithSpaces() {
00092:         Scanner console = new Scanner(" " + NEWLINE + "validkey");
00093:         String expResult = "validkey";
00094:         String result = Cipher.getKey(console);
00095:         assertEquals(expResult, result);
00096:     }
00097:
00098:     /**
00099:      * Checks getKey for invalid key as Unicode input.
```

```
00100:      */
00101:     @Test
00102:     public void testGetKeyWithUnicodeKeysFirst() {
00103:         Scanner console = new Scanner("\u00DF" + NEWLINE + "validkey");
00104:         String expResult = "validkey";
00105:         String result = Cipher.getKey(console);
00106:         assertEquals(expResult, result);
00107:     }
00108:
00109:     /**
00110:      * Checks getKey for invalid key as capital letters input.
00111:      */
00112:     @Test
00113:     public void testGetKeyWithCapitalLetter() {
00114:         Scanner console = new Scanner("Hi" + NEWLINE + "validkey");
00115:         String expResult = "validkey";
00116:         String result = Cipher.getKey(console);
00117:         assertEquals(expResult, result);
00118:     }
00119:
00120:     /**
00121:      * Checks getKey for invalid key as space in middle as input.
00122:      */
00123:     @Test
00124:     public void testGetKeyWithSpaceInMiddle() {
00125:         Scanner console = new Scanner("hi bro" + NEWLINE + "validkey");
00126:         String expResult = "validkey";
00127:         String result = Cipher.getKey(console);
00128:         assertEquals(expResult, result);
00129:     }
00130:
00131:     /**
00132:      * Test for input as valid file.
```

```
00133:      */
00134:     @Test
00135:     public void testGetInputScannerWithValidFile() {
00136:         String goodFileName = "TaleOfTwoCities.txt";
00137:         File goodFile = new File(goodFileName);
00138:         Scanner mockConsole = null;
00139:
00140:         if (goodFile.exists()) {
00141:             mockConsole = new Scanner(goodFileName); //this will act like the user input
00142:             Scanner fileScanner = Cipher.getInputScanner(mockConsole);
00143:             Scanner taleScanner = new Scanner(taleOriginal);
00144:             assertEquals(fileScanner.nextLine(), taleScanner.nextLine()); //this isn't an exhaustive test, but is a start
00145:         }
00146:     }
00147:
00148:     /**
00149:      * Test for input as invalid file.
00150:      */
00151:     @Test
00152:     public void testGetInputScannerWithInvalidFilesFirst() {
00153:         String goodFileName = "TaleOfTwoCities.txt";
00154:         String badFileName = "aux.txt"; //anything with AUX or PRN is not allowed in filenames on Windows systems. It's a relatively safe bet that a file with
this name will not exist on most systems.
00155:         File goodFile = new File(goodFileName);
00156:         Scanner mockConsole = null;
00157:
00158:         if (goodFile.exists()) {
00159:             mockConsole = new Scanner(badFileName + NEWLINE + goodFileName); //this will act like the user input
00160:             Scanner fileScanner = Cipher.getInputScanner(mockConsole);
00161:             Scanner taleScanner = new Scanner(taleOriginal);
00162:             assertEquals(fileScanner.nextLine(), taleScanner.nextLine());
00163:         }
00164:     }
```

```
00165:
00166:     /**
00167:      * Processes encryption in proccesFile method with EncrpytTale.
00168:      */
00169:     @Test
00170:     public void testProcessFileEncryptTale() {
00171:         String key = "fun";
00172:         Scanner input = new Scanner(taleOriginal);
00173:         ByteArrayOutputStream byteOutStream = new ByteArrayOutputStream();
00174:         PrintStream output = new PrintStream(byteOutStream);
00175:         Cipher.processFile(true, key, input, output);
00176:         String allWrittenLines = byteOutStream.toString();
00177:         assertTrue(allWrittenLines.contains(taleEncrypted));
00178:     }
00179:
00180:     /**
00181:      * Processes decryption in proccesFile method with EncrpytTale.
00182:      */
00183:     @Test
00184:     public void testProcessFileDecryptTale() {
00185:         String key = "fun";
00186:         Scanner input = new Scanner(taleEncrypted);
00187:         ByteArrayOutputStream byteOutStream = new ByteArrayOutputStream();
00188:         PrintStream output = new PrintStream(byteOutStream);
00189:         Cipher.processFile(false, key, input, output);
00190:         String allWrittenLines = byteOutStream.toString();
00191:         assertTrue(allWrittenLines.contains(taleOriginal));
00192:     }
00193:
00194:     /**
00195:      * Test input with numbers and has key as cats.
00196:      */
00197:     @Test
```

```
00198:     public void testEncryptLine1() {
00199:         String expected = "Oexl ct 102 Xdo Sm.";
00200:         assertEquals(expected, Cipher.encryptLine("cats", "Meet at 102 Elm St."));
00201:     }
00202:
00203:     /**
00204:      * Test input with passage from encrypt tales with new lines and has key as
00205:      * fun.
00206:      */
00207:     @Test
00208:     public void testEncryptLine2() {
00209:         String input = "It was the best of times," + NEWLINE
00210:                 + "it was the worst of times," + NEWLINE;
00211:         String expected = "Nn jfm gmy ojmg tz gngrx," + NEWLINE
00212:                 + "nn jfm gmy jtlfy is yczjm," + NEWLINE;
00213:         String actual = Cipher.encryptLine("fun", input);
00214:         assertEquals(expected, actual);
00215:     }
00216:
00217:     /**
00218:      * Test input with random words and has key as "bbb" for making sure all
00219:      * chars move same.
00220:      */
00221:     @Test
00222:     public void testEncryptLine3() {
00223:         String input = "Hello human";
00224:         String expected = "Ifmmp ivnbo";
00225:         String actual = Cipher.encryptLine("bbb", input);
00226:         assertEquals(expected, actual);
00227:     }
00228:
00229:     /**
00230:      * Test input with words with symbols and has key as cool.
```

```
00231:      */
00232:     @Test
00233:     public void testEncryptLine4() {
00234:         String input = "T!h@i#s i$s c%o^o*l";
00235:         String expected = "V!v@w#d k$g q%z^q*z";
00236:         String actual = Cipher.encryptLine("cool", input);
00237:         assertEquals(expected, actual);
00238:     }
00239:
00240:     /**
00241:      * Test input with spaces and has key as wow.
00242:      */
00243:     @Test
00244:     public void testEncryptLine5() {
00245:         String input = "   Woa h";
00246:         String expected = "   Scw d";
00247:         String actual = Cipher.encryptLine("wow", input);
00248:         assertEquals(expected, actual);
00249:     }
00250:
00251:     /**
00252:      * Test input with capital letters and has key as grade.
00253:      */
00254:     @Test
00255:     public void testEncryptLine6() {
00256:         String input = "I JuST wAnT aN A";
00257:         String expected = "O AuVX cRnW eT R";
00258:         String actual = Cipher.encryptLine("grade", input);
00259:         assertEquals(expected, actual);
00260:     }
00261:
00262:     /**
00263:      * Test input with symbols only and has key as fun.
```

```
00264:      */
00265:     @Test
00266:     public void testEncryptLine7() {
00267:         String input = "!//
00268:         String expected = "!//
00269:         String actual = Cipher.encryptLine("fun", input);
00270:         assertEquals(expected, actual);
00271:     }
00272:
00273:     /**
00274:      * Test input with numbers and has key as cats.
00275:      */
00276:     @Test
00277:     public void testDecryptLine1() {
00278:         String expected = "Meet at 102 Elm St.";
00279:         assertEquals(expected, Cipher.decryptLine("cats", "Oexl ct 102 Xdo Sm."));
00280:     }
00281:
00282:     /**
00283:      * Test input with passage from encrypt tales with new lines and has key as
00284:      * fun.
00285:      */
00286:     @Test
00287:     public void testDecryptLine2() {
00288:         String input = "Nn jfm gmy ojmg tz gngrx," + NEWLINE
00289:                 + "nn jfm gmy jtlfy is yczjm," + NEWLINE;
00290:         String expected = "It was the best of times," + NEWLINE
00291:                 + "it was the worst of times," + NEWLINE;
00292:         String actual = Cipher.decryptLine("fun", input);
00293:         assertEquals(expected, actual);
00294:     }
00295:
00296:     /**
```

```
00297:      * Test input with random words and has key as "bbb" for making sure all
00298:      * chars move same.
00299:      */
00300:     @Test
00301:     public void testDecryptLine3() {
00302:         String input = "Ifmmp ivnbo";
00303:         String expected = "Hello human";
00304:         String actual = Cipher.decryptLine("bbb", input);
00305:         assertEquals(expected, actual);
00306:     }
00307:
00308:     /**
00309:      * Test input with words with symbols and has key as cool.
00310:      */
00311:     @Test
00312:     public void testDecryptLine4() {
00313:         String input = "V!v@w#d k$g q%z^q*z";
00314:         String expected = "T!h@i#s i$s c%o^o*l";
00315:         String actual = Cipher.decryptLine("cool", input);
00316:         assertEquals(expected, actual);
00317:     }
00318:
00319:     /**
00320:      * Test input with spaces and has key as wow.
00321:      */
00322:     @Test
00323:     public void testDecryptLine5() {
00324:         String input = "   Scw d";
00325:         String expected = "   Woa h";
00326:         String actual = Cipher.decryptLine("wow", input);
00327:         assertEquals(expected, actual);
00328:     }
00329:
```

```
00330:     /**
00331:      * Test input with capital letters and has key as grade.
00332:      */
00333:     @Test
00334:     public void testDecryptLine6() {
00335:         String input = "O AuVX cRnW eT R";
00336:         String expected = "I JuST wAnT aN A";
00337:         String actual = Cipher.decryptLine("grade", input);
00338:         assertEquals(expected, actual);
00339:     }
00340:
00341:     /**
00342:      * Test input with symbols only and has key as fun.
00343:      */
00344:     @Test
00345:     public void tesDecryptLine7() {
00346:         String input = "!//
00347:         String expected = "!//
00348:         String actual = Cipher.decryptLine("fun", input);
00349:         assertEquals(expected, actual);
00350:     }
00351: }
```