```
/*
 * Open Source Physics software is free software as described near the bottom of this code file.
 *
 * For additional information and documentation on Open Source Physics please see:
 * <http://www.opensourcephysics.org/>
 */

package org.opensourcephysics.controls;
import java.security.spec.AlgorithmParameterSpec;
import java.security.spec.KeySpec;
import javax.crypto.Cipher;
import javax.crypto.SecretKey;
import javax.crypto.SecretKeyFactory;
import javax.crypto.spec.PBEKeySpec;
import javax.crypto.spec.PBEParameterSpec;

/**
 * A class to represent an encrypted version of a UTF-8-encoded String.
 *
 * @author Doug Brown
 * @version 1.0 May 2006
 */
public class Cryptic {
  // static fields
  private static String encoding = "UTF-8";             //$NON-NLS-1$
  private static String keyFormat = "PBEWithMD5AndDES"; //$NON-NLS-1$
  private static byte[] salt = {(byte) 0x09, (byte) 0x9C, (byte) 0xC8, (byte) 0x23, (byte) 0x1E, (byte) 0xAA, (byte) 0x
  private static int interactions = 19;
  private static final String DEFAULT_PW = "ospWCMBACBJDB"; //$NON-NLS-1$
  // instance fields
  private String cryptic;                               // the incrypted form of the input

  /**
   * Protected no-arg constructor has null cryptic.
   */
  protected Cryptic() {

  /** empty block */
  }

  /**
   * Public constructor with input string.
   *
   * @param input UTF-8 String to encrypt
   */
  public Cryptic(String input) {
    encrypt(input);
  }

  /**
   * Public constructor with input and password.
   *
   * @param input UTF-8 String to encrypt
   * @param password
   */
  public Cryptic(String input, String password) {
    encrypt(input, password);
  }

  /**
   * Encrypts the input and saves in cryptic form.
   *
   * @param input UTF-8 String to encrypt
   * @return the encrypted content
   */
```

```
66    public String encrypt(String content) {
67      return encrypt(content, DEFAULT_PW);
68    }
69
70    /**
71     * Encrypts the input with a password and saves in cryptic form.
72     *
73     * @param input UTF-8 String to encrypt
74     * @return the encrypted content
75     */
76    public String encrypt(String content, String password) {
77      try {
78        // create the key and parameter spec
79        KeySpec keySpec = new PBEKeySpec(password.toCharArray(), salt, interactions);
80        SecretKey key = SecretKeyFactory.getInstance(keyFormat).generateSecret(keySpec);
81        AlgorithmParameterSpec paramSpec = new PBEParameterSpec(salt, interactions);
82        // create the cipher
83        Cipher ecipher = Cipher.getInstance(key.getAlgorithm());
84        ecipher.init(Cipher.ENCRYPT_MODE, key, paramSpec);
85        // get byte[] from content and encrypt with cipher
86        byte[] bytes = content.getBytes(encoding);
87        byte[] enc = ecipher.doFinal(bytes);
88        // save encrypted bytes as string of chars 0-63
89        // note this doubles the string length
90        cryptic = new String(Base64Coder.encode(enc));
91      } catch(Exception ex) {
92        ex.printStackTrace();
93      }
94      return cryptic;
95    }
96
97    /**
98     * Gets the decrypted string.
99     * @return the decrypted string
100    */
101   public String decrypt() {
102     return decrypt(DEFAULT_PW);
103   }
104
105   /**
106    * Gets the decrypted string using a password.
107    *
108    * @param password the password
109    * @return the decrypted string
110    */
111   public String decrypt(String password) {
112     try {
113       // create the key and parameter spec
114       KeySpec keySpec = new PBEKeySpec(password.toCharArray(), salt, interactions);
115       SecretKey key = SecretKeyFactory.getInstance(keyFormat).generateSecret(keySpec);
116       AlgorithmParameterSpec paramSpec = new PBEParameterSpec(salt, interactions);
117       // create the cipher
118       Cipher dcipher = Cipher.getInstance(key.getAlgorithm());
119       dcipher.init(Cipher.DECRYPT_MODE, key, paramSpec);
120       byte[] dec = null;
121       try {
122         dec = Base64Coder.decode(cryptic);
123       } catch(IllegalArgumentException ex) {
124         // decode legacy files encoded with sun encoder
125         dec = new sun.misc.BASE64Decoder().decodeBuffer(cryptic);
126       }
127       byte[] bytes = dcipher.doFinal(dec);
128       return new String(bytes, encoding);
129     } catch(Exception ex) {
130       ex.printStackTrace();
131     }
132     return null;
133   }
134
135   /**
136    * Gets the cryptic.
137    * @return the encrypted version of the input
138    */
139   public String getCryptic() {
140     return cryptic;
141   }
142
143   /**
144    * Sets the cryptic.
145    * @param encrypted an encrypted string
146    */
147   public void setCryptic(String encrypted) {
148     cryptic = encrypted;
```

```
  }

  /**
   * Returns an ObjectLoader to save and load data for this class.
   *
   * @return the object loader
   */
  public static XML.ObjectLoader getLoader() {
    return new Loader();
  }

  /**
   * A class to save and load data for this class.
   */
  static class Loader implements XML.ObjectLoader {
    /**
     * Saves an object's data to an XMLControl.
     *
     * @param control the control to save to
     * @param obj the object to save
     */
    public void saveObject(XMLControl control, Object obj) {
      Cryptic cryptic = (Cryptic) obj;
      control.setValue("cryptic", cryptic.getCryptic()); //$NON-NLS-1$
    }

    /**
     * Creates a new object.
     *
     * @param control the control
     * @return the newly created object
     */
    public Object createObject(XMLControl control) {
      return new Cryptic();
    }

    /**
     * Loads an object with data from an XMLControl.
     *
     * @param control the control
     * @param obj the object
     * @return the loaded object
     */
    public Object loadObject(XMLControl control, Object obj) {
      Cryptic cryptic = (Cryptic) obj;
      cryptic.setCryptic(control.getString("cryptic")); //$NON-NLS-1$
      return obj;
    }

  }

}

/*
 * Open Source Physics software is free software; you can redistribute
 * it and/or modify it under the terms of the GNU General Public License (GPL) as
 * published by the Free Software Foundation; either version 2 of the License,
 * or(at your option) any later version.

 * Code that uses any portion of the code in the org.opensourcephysics package
 * or any subpackage (subdirectory) of this package must must also be be released
 * under the GNU GPL license.
 *
 * This software is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston MA 02111-1307 USA
 * or view the license online at http://www.gnu.org/copyleft/gpl.html
 *
 * Copyright (c) 2007  The Open Source Physics project
 *                     http://www.opensourcephysics.org
 */
```