Encrypting and decrypting in Java ME

This code snippet shows how you can encrypt and decrypt text strings with Java ME, using methods of the `Cipher` object.

Source file: EnDecryptingMIDlet.java

```
    /**
     * Executes the snippet.
     * Implementation executes the cryptingDecrypting() method with
     * "Test string!" parameter.
     */
    private void executeSnippet() {
        String text = "Snippet executed...";
        printString(text);

        cryptingDecrypting("Test string!");
        try {
            logPrintString(text);
        } catch (IOException ex) {
            printString(ex.getMessage());
        }
    }

    /**
     * This function encrypts and decrypts the input parameter string
     * @param textToCrypting - string that will be encrypted and decrypted
     */
    protected void cryptingDecrypting(String textToCrypting) {
        //Object of this class provides the functionality for
        //encryption and decryption.
        Cipher cipher;

        try {
            //parameter "DES" specifies type of cipher we want to create
            //through the factory method. It includes algorithm, mode and
            //padding. You can define only algorithm and in that case default
            //values will be used for mode and padding.
            cipher = Cipher.getInstance("DES");
        } catch (NoSuchAlgorithmException ex) {
            //requested cryptographic algorithm is not available
            printString(ex.toString());
            printString("Error! Snippets execution stopped.");
            return;
        } catch (NoSuchPaddingException ex) {
            //requested padding mechanism is not available in the environment.
            printString(ex.toString());
            printString("Error! Snippets execution stopped.");
            return;
        }

        //The lenght of keyString is 8. It's important characteristic
        //for encryption
        String keyString = "testtest";

        byte[] keyData = keyString.getBytes();

        //key object specifies a secret key
        //it uses to construct the secret key for our cipher object from a byte
        //array
        SecretKeySpec key = new SecretKeySpec(keyData, 0, keyData.length,
                "DES");

        try {
            //initializing with setting up the type of operation it's to be
            //used for. In this case - for encryption.
            cipher.init(Cipher.ENCRYPT_MODE, key);
        } catch (InvalidKeyException ex) {
            //given key object is inappropriate for initializing this cipher
            printString(ex.toString());
            printString("Error! Snippets execution stopped.");
            return;
        }

        int cypheredBytes = 0;

        printString("Source string: " + textToCrypting);

        byte[] inputBytes = textToCrypting.getBytes();
        byte[] outputBytes = new byte[100];

        try {
            //doFinal method encrypts data to outputBytes array.
            //for unknown or big counts of data update method more recomended
            //counts of crypted bytes saved inside cypheredBytes variable
            cypheredBytes = cipher.doFinal(inputBytes, 0, inputBytes.length,
                    outputBytes, 0);
        } catch (IllegalStateException ex) {
            printString(ex.toString());
            printString("Error! Snippets execution stopped.");
            return;
        } catch (ShortBufferException ex) {
            printString(ex.toString());
            printString("Error! Snippets execution stopped.");
            return;
        } catch (IllegalBlockSizeException ex) {
            printString(ex.toString());
            printString("Error! Snippets execution stopped.");
            return;
        } catch (BadPaddingException ex) {
            printString(ex.toString());
            printString("Error! Snippets execution stopped.");
            return;
        }

        String str = new String(outputBytes, 0, cypheredBytes);
        inputBytes = str.getBytes();
```


(C) Copyright Nokia 2013. All rights reserved.

```
        printString("Crypted string:" + str);

        try {
            //change state of the cipher object for decrypting
            cipher.init(Cipher.DECRYPT_MODE, key);
        } catch (InvalidKeyException ex) {
            printString(ex.toString());
            printString("Error! Snippets execution stopped.");
            return;
        }

        try {
            //decrypts data
            cypheredBytes = cipher.doFinal(inputBytes, 0, inputBytes.length,
                      outputBytes, 0);
        } catch (IllegalStateException ex) {
            printString(ex.toString());
            printString("Error! Snippets execution stopped.");
            return;
        } catch (ShortBufferException ex) {
            printString(ex.toString());
            printString("Error! Snippets execution stopped.");
            return;
        } catch (IllegalBlockSizeException ex) {
            printString(ex.toString());
            printString("Error! Snippets execution stopped.");
            return;
        } catch (BadPaddingException ex) {
            printString(ex.toString());
            printString("Error! Snippets execution stopped.");
            return;
        }

        str = new String(outputBytes, 0, cypheredBytes);
        printString("Decrypted string:" + str);
        printString(" Ok!   " + str);
    }
```

Postconditions

After choosing the command `Execute snippet`, the MIDlet displays 'Test string!' in the text field. This message is then encrypted, and the encrypted message is displayed. Finally, the encrypted message is decrypted, and the decrypted message is displayed.

Supplementary material

This code snippet is part of the stub concept, which means that it has been patched on top of a template application in order to be more useful for developers. The version of the Java ME stub application used as a template in this snippet is v1.1.

- The patched, executable application that can be used to test the features described in this snippet is available for download at Media:EncryptingAndDecrypting.zip.
- You can view all the changes that are required to implement the above-mentioned features. The changes are provided in unified diff and colour-coded diff (HTML) formats in Media:EncryptingAndDecrypting.diff.zip.
- For general information on applying the patch, see Using Diffs.
- For unpatched stub applications, see Example app stubs with logging framework.

(C) Copyright Nokia 2013. All rights reserved.

(C) Copyright Nokia 2013. All rights reserved.