```
package com.mycompany.cryptodummy;
// Classe "CryptoDummy.java"
import java.io.*;
public class CryptoDummy
{   private  byte[] textoCifrado;
    private  byte[] textoDecifrado;
    public CryptoDummy()
  { textoCifrado = null;
    textoDecifrado = null;
  }
  public void geraChave(File fDummy)throws IOException
  { // Gera uma chave Dummy simetrica (dk: 0 a 100):
  int dk = (int) (Math.random()*101);
// Grava a chave Dumyy simetrica em formato serializado
ObjectOutputStream oos = new ObjectOutputStream(new
FileOutputStream(fDummy));
oos.writeObject(dk);
oos.close();

  }
  public void geraCifra(byte[] texto, File fDummy)
        throws IOException, ClassNotFoundException
  { ObjectInputStream ois = new ObjectInputStream (new FileInputStream (fDummy));
  int iDummy = (Integer) ois.readObject();
  ois.close();
  textoCifrado = texto;
  for(int i = 0; i < texto.length; i++)
```

```
 { textoCifrado[i] = (byte) (textoCifrado[i] + i + iDummy);
 }
 }

 public byte[] getTextoCifrado() throws Exception
 { return textoCifrado;
 }
 public void geraDecifra(byte[] texto, File fDummy)
       throws IOException, ClassNotFoundException
 { ObjectInputStream ois = new ObjectInputStream (new FileInputStream (fDummy));
 int iDummy = (Integer) ois.readObject();
 ois.close();
 textoDecifrado = texto;
 for(int i = 0; i < texto.length; i++)
 { textoDecifrado[i] = (byte) (textoDecifrado[i] - i - iDummy);
       }
 }

 public byte[] getTextoDecifrado() throws Exception
 { return textoDecifrado;

 }
}



 // Classe "CryptoAES.java"
import java.io.*;
```

```
import javax.crypto.*;
import javax.crypto.spec.*;
import java.security.*;
import java.security.cert.*;
public class CryptoAES
{ private byte[] textoCifrado;
private byte[] textoDecifrado;
public CryptoAES()
{textoCifrado = null;
textoDecifrado = null;
}
public void geraChave(File fSim)
     throws IOException, NoSuchAlgorithmException,
InvalidAlgorithmParameterException, CertificateException, KeyStoreException
{ // Gera uma chave simetrica de 128 bits:
 KeyGenerator kg =  KeyGenerator.getInstance("AES");
 kg.init(128);
 SecretKey sk = kg.generateKey();
 // Grava a chave simetrica em formato serializado
 ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream(fSim));
 oos.writeObject(sk);
 oos.close();
}
public void geraCifra(byte[] texto, File fSim)
     throws NoSuchAlgorithmException, NoSuchPaddingException,
InvalidKeyException, IllegalBlockSizeException, BadPaddingException,
InvalidAlgorithmParameterException, IOException, ClassNotFoundException

{ ObjectInputStream ois = new ObjectInputStream (new FileInputStream (fSim));
```

```
SecretKey iSim = (SecretKey) ois.readObject();
byte[] chave = iSim.getEncoded();
ois.close();
Cipher aescf = Cipher.getInstance ("AES/CBC/PKCS5Padding");
IvParameterSpec ivspec = new IvParameterSpec (new byte[16]);
aescf.init (Cipher.ENCRYPT_MODE, new SecretKeySpec (chave, "AES"), ivspec);
textoCifrado = aescf.doFinal (texto);
}
public byte[] getTextoCifrado() throws Exception
{ return textoCifrado;
}
public void geraDecifra(byte[] texto, File fSim)
    throws NoSuchAlgorithmException, NoSuchPaddingException,
InvalidKeyException,
    IllegalBlockSizeException, BadPaddingException,
    InvalidAlgorithmParameterException, IOException, ClassNotFoundException
{ ObjectInputStream ois = new ObjectInputStream (new FileInputStream (fSim));
SecretKeySpec iSim = (SecretKeySpec) ois.readObject();
ois.close();
  Cipher aescf =  Cipher.getInstance ("AES/CBC/PKCS5Padding");
  IvParameterSpec ivspec = new IvParameterSpec (new byte[16]);
  aescf.init (Cipher.DECRYPT_MODE, iSim, ivspec);
  textoDecifrado = aescf.doFinal (texto);
}
public byte[] getTextoDecifrado() throws Exception
{ return textoDecifrado;
}
}
```

```
      // Classe "Impressora.java"
public class Impressora
{ public String hexBytesToString(byte[] b)
{ String sOut = "";
  String sBgn = "";
  String sMdl = "";
  String sEnd = "";
  String sSpc = "                                        "; // 48 espaços
for(int i = 0; i < b.length; i++)
{ // A cada linha de 16 bytes hexadecimas faz:
   if(i%16==0) sBgn += Integer.toHexString(i&0xFFFF | 0x10000).substring(1,5) + " = ";
  // Monta a String do meio, contendo os bytes lidos
sMdl += Integer.toHexString(b[i] & 0xFF | 0X100).substring(1,3) + " ";
  // Monta a String do final, contendo os carecteres lidos
  if(b[i] >+ 32 && b[i] <+ 126) sEnd+= (char) b[i];
  else sEnd += ".";
  //Monta linha a cada 16 caracteres lidos
  if((i % 16 == 15)  || (i == b.length - 1))
  { sOut += sBgn+sMdl+sSpc.substring(3*((i%16)+1),sSpc.length())+" - "+sEnd+"\n";
  sBgn = sMdl = sEnd = "";
   }

  }
return sOut;

}
```

```
}
// Classe "TesteCrypto.java"
import java.io.File;
public class TesteCrypto
{ public static void main(String[] args) throws Exception
{ String sMsgClara = "Oi, alunos do IMT!";
 String sMsgCifrada = null;
 String sMsgDecifrada = null;
 byte[] bMsgClara = null;
 byte[] bMsgCifrada = null;
 byte[] bMsgDecifrada = null;
 //Instancia objeto da classe impressora
 Impressora prn = new Impressora();
 //Imprime marcador de bloco
 System.out.println("''''''''''''''''''''''''''''''''''''''''''''''''''");
 //Imprime texto
 System.out.println(">>> Imprimindo mensagem original...");
 System.out .println("");
 // Converte o texto Strinf dado no equivalente byte []
 bMsgClara = sMsgClara.getBytes("ISO-8859-1");
 //Imprime o cabecalho da mensagem
 System.out.println("Mensagem Clara (Hexadecimal):");
 //Imprime o texto original em Hexadecimal
 System.out.print(prn.hexBytesToString(bMsgClara));
 System.out.println("");
 // Imprime cabecalho da mensagem
 System.out.println("Mensagem Clara (String):");
```

```
 // Imprime o texto original em String
 System.out.println(sMsgClara);
 System.out.println("");
 /*
  *
  *Criptografia Dumyy ---------------------------------------------
 */
//Imprime Texto
System.out.println(">>> Cifrando com o algoritmo Dummy...");
System.out.println("");
//Instancia um objeto da classee CryptoDummy
CryptoDummy cdummy = new CryptoDummy();
//Gera a chave criptografica Dummy simetrica e nome do arquivo onde sera armazenada
cdummy.geraCifra(bMsgClara, new File ("chave.dummy"));



}


}
```