1. Hash collision is: x and y are distinct values, yet when input into hash function H, they product the same output.
   a. **H(x) = x mod 7^{12}, where x can be any integer.**
      So, 1 and ($1+7^{12}$) would have a hash collision, because 1 mod 7^{12} is equal to ($1+7^{12}$) mod 7^{12}.
   b. **H(x) = # of 1-bits in x, where x can be any bit string**
      If x is an 8-bit binary integer, if x=00001101, H(x) returns 3, so H(x)=3; if x=00110100, H(x returns 3, so H(x)=3. These two H(x) are equal, and that will cause hash collision.
   c. **H(x)=the three least significant bits of x, where x can be any bit string**
      If x is an 8-bit binary integer $d_1d_2d_3d_4d_5d_6d_7d_8$, if x=10011110, so H(x)=110, if x=11010110, so H(x)=110. These two H(x) are equal, and that will cause hash collision.

2. Prove the statement: In a class of 500 students, there must be two students with the same birthday.

   Because one year has 366 days at most, so if a class has 500 students, there must be two students with the same birthday. And by the pigeonhole principle, the probability reaches 100% when the number of students reaches 367.

   Pigeonholes: Each day in a year can be a birthday, and one year has 366 days at most [1-366]
   Pigeons: Student [1-500]
   Collision: There must be two students "mapped" to a specific birthday.

3. Find an x such that H (x ◦ id) ∈ Y where
   a. H = SHA-256
   b. id = 0xED00AF5F774E4135E7746419FEB65DE8AE17D6950C95CEC3891070FBB5B03C77
   c. Y is the set of all 256 bit values that have some byte with the value 0x1D.
   Assume SHA-256 is puzzle-friendly. Your answer for x must be in hexadecimal. You may provide your code for partial credit, if your x value is incorrect. You may use the accompanying CryptoReference1.java file to help you with this question.

   One of the value of x:
   C897FAE5F1811531CAE1D5E1A702ECBFA60613A309E9E0645FDA6EA32F4925DD

   (The code is attached below.)

4. Alice and Bob want to play a game over SMS text where Alice chooses a number between 1 and 10 in her head, and then Bob tries to guess that number. If Bob guesses correctly, he wins. Otherwise, Alice wins. However, Bob complains that the game isn't fair, because even if Bob guessed correctly, Alice could lie and claim that she chose a different number than what she initially chose. What can Alice do to prove that she didn't change the number she initially chose? Devise a mechanism to address Bob's concern. Provide a detailed explanation of the mechanism and why it works. An answer with insufficient detail will not receive credit.
   We can get Alice and Bob to use a hash function as commitment, like SHA-256 or others. Then Alice choose a random value between 1 and 10. Then takes a secure hash of that random value, including with salt value, and send it to Bob.
   Next Bob send the hash value of the number he guesses to Alice, and Alice send the salt value to Bob.
   If Alice verify the value Bob sent plus the salt value is equal to hers, and Bob use the hash function to verify the value he guesses plus the salt value is equal to the value Alice sent, Bob will win, otherwise, Alice will win.

   Because sha256 is collision resistant so Alice would not be able to find two numbers that have the same hash. And Bob would not cheat because you include a nonce when Alice generates her hash. This means that Bob will not be able to reverse the hash received from Alice

```
/*

 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package Homework1;

import java.io.ByteArrayOutputStream;
import java.io.IOException;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.util.Random;
import javax.xml.bind.DatatypeConverter;

/**
 *
 * @author zhaos
 */
public class HomeWork1 {

   public static void main(String[] args) throws NoSuchAlgorithmException, IOException {

      String ID = "ED00AF5F774E4135E7746419FEB65DE8AE17D6950C95CEC3891070FBB5B03C77";
      byte[] b = DatatypeConverter.parseHexBinary(ID);
    //String        test=
          "6642ADC718517EA646E44BF1CF5E4050B45045F5EF2A213CF8E38850F5D85E20";
   //byte[] testByte = DatatypeConverter.parseHexBinary(test);
      for (int count = 1; count < 100; count++) {

         byte[] x = new byte[32]; //256 bit array
         new Random().nextBytes(x); //pseudo-random
         String xHex = DatatypeConverter.printHexBinary(x);//the x

         ByteArrayOutputStream outputStream = new ByteArrayOutputStream();
         outputStream.write(x);
         outputStream.write(b);
         byte concat[] = outputStream.toByteArray();
         String concatHex = DatatypeConverter.printHexBinary(concat);

         MessageDigest digest = MessageDigest.getInstance("SHA-256");
```

```
        byte[] hashResult = digest.digest(concat); //SHA256 hash
        String finalHash = DatatypeConverter.printHexBinary(hashResult);

        for (byte bt : hashResult) {
           if (bt == 0x1D) {
//                for (int j = 0; j < hashResult.length; j++) {
//                   System.out.print(hashResult[j]);
//                }
              System.out.println();
              System.out.println("the value of x :" + xHex);
              System.out.println("x ◦ id: " + concatHex);
              System.out.println("Hash Result : " + finalHash);
           }
        }
     }
  }
}
```