```
/*

 "Abstract Names"

 v. 0.3.3

 ©Jonn Esternon / Cireaka

 */

/*
 Programmer's Introspections:

 Will I focus on just phonetics?
 Or should I include decorative spellings
 of phonemes?

 What if, we try to generate the pronunciations first,
 and then, along that, generate the possible spellings?
 That would be above my skill set.
 Generating a spelling right away
 would be straightforward.

 How about the African click sound, tsk tsk tsk? Perhaps no.

 How about the beautiful effect of repetition?
 (As in the name, Parangaongao)

 Something (an int) that will monitor the
 current position of the next phoneme.
 (e.g. i need to make gh[f] (as in rough)
 to not show up as the first phoneme,
 that won't be aesthetic if i let it.)
 \\ generatePhoneme(position, phoneme); \\

 How 'bout a random dash in between syllables?
 e.g. Bar-abbas

 In future developments, the user will be able to
 choose between different algorithms.
 They can choose the three-syllabic algo in version .01 and others.
 */

import java.util.*;
import java.util.Random;

public class Main{
    //main method
    public Random r = new Random();
    public static void main(String[] args){



        abstractNames();
        trademark();

        //

        algorithm01(12);
        algorithm02(3);
        //algorithmChaos(1);

        //

        cireaka();
    }

    static void abstractNames(){
        System.out.println("ABSTRACT NAMES\n");
    }

    static void trademark(){
        System.out.println("This code generates nonsense.\n");
    }

    static void cireaka(){
        System.out.println("\n\n\n\n©Jonn David Esternon / Cireaka");
    }

    //methods...

    static void algorithm01(int num){
        Random r = new Random();
        System.out.println("\nAlgorithm 01:\n");

        String[] vowel =
        {
            "a", "e", "i", "o", "u",
        };
        String[] consonant =
        {
            "b", "c", "ch", "d", "f", "ph", "g", "h", "j", "k", "l", "m",
            "n", "ng", "p", "q", "r", "s", "sh", "t", "th", "v", "w", "x", "y", "z",
        };
        String[] vowelCap =
        {
            "A", "E", "I", "O", "U",
        };
        String[] consonantCap =
        {
            "B", "C", "Ch", "D", "F", "Ph", "G", "H", "J", "K", "L", "M",
            "N", "Ng", "P", "Q", "R", "S", "Sh", "T", "Th", "V", "W", "X", "Y", "Z",
        };
        String[] dipthong =
        {
            "ae", "ai", "ao", "au",
            "ea", "ee", "ei", "eo", "eu",
            "ia", "ie", "io", "iu",
            "oa", "oe", "oi", "oo", "ou",
            "ua", "ue", "ui", "uo",
                                  "y"
        };
        String[] dipthongCap =
        {
            "Ae", "Ai", "Ao", "Au",
            "Ea", "Ee", "Ei", "Eo", "Eu",
            "Ia", "Ie", "Io", "Iu",
            "Oa", "Oe", "Oi", "Oo", "Ou",
            "Ua", "Ue", "Ui", "Uo",
                                  "Y"
        };

        for(int i = 0; i < num; i ++){
            boolean consonance = r.nextBoolean();
            //consonance = true;
            int sets = r.nextInt(5);
            if(sets < 2){
                sets += 2;
            }
            //System.out.println("consonance of the first letter: " + consonance);

            String[] c = {
```

```
    consonant[r.nextInt(consonant.length)],
    consonant[r.nextInt(consonant.length)],
    consonant[r.nextInt(consonant.length)],
    consonant[r.nextInt(consonant.length)],
    consonant[r.nextInt(consonant.length)],
    consonant[r.nextInt(consonant.length)],
};
String[] v = {
    vowel[r.nextInt(vowel.length)],
    vowel[r.nextInt(vowel.length)],
    vowel[r.nextInt(vowel.length)],
    vowel[r.nextInt(vowel.length)],
    vowel[r.nextInt(vowel.length)],
    vowel[r.nextInt(vowel.length)],
};
//
if(consonance == true){
    c[0] = consonantCap[r.nextInt(consonantCap.length)];
} else if(consonance == false){
    v[0] = vowelCap[r.nextInt(vowelCap.length)];
}
//
boolean dipthongize = r.nextBoolean();
//System.out.println("is there a decor vowel: " + decor);
if(dipthongize == true){
    int decorPosition = r.nextInt(sets);
    //System.out.println("decor pos: " + decorPosition);
    if(decorPosition == 0 && consonance == false){
        v [0] = dipthongCap[r.nextInt(dipthongCap.length)];
    } else{
        v[decorPosition] = dipthong[r.nextInt(dipthong.length)];
    }
}
// Repetition
                boolean repeatC = false;
                int cRepZ = r.nextInt(100);
                if(cRepZ < 30){
                        repeatC = true;
                }
                int c2repeat = r.nextInt(sets-1);
                if(consonance == true && c2repeat ==0){
                        c2repeat += 1;
                }
                if(repeatC == true){
                        c[c2repeat + 1] = c[c2repeat];
                        //System.out.println("cRepeated: " + c[c2repeat + 1]);
                }

                boolean repeatV = r.nextBoolean();
                int vRepZ = r.nextInt(100);
                if(vRepZ < 30){
                        repeatV = true;
                }
                int v2repeat = r.nextInt(sets);
                if(consonance == false && v2repeat == 0){
                        v2repeat += 1;
                }
                if(repeatV == true){
                        v[v2repeat + 1] = v[v2repeat];
                        //System.out.println("vRepeated: " + v[v2repeat + 1]);
                }

                //
                boolean lastDoExist = r.nextBoolean();

        if(consonance == true){
            for(int j = 0; j < sets; j++){
                System.out.print(c[j]);
                if(j == sets-1 && lastDoExist == true){
                    v[j] = (" ");
                }
                System.out.print(v[j]);
            }
            System.out.print("\n");
        } else if(consonance == false){
            for(int j = 0; j < sets; j++){
                System.out.print(v[j]);
                if(j == sets-1 && lastDoExist == true){
                    c[j] = (" ");
                }
                System.out.print(c[j]);
            }
            System.out.print("\n");
        }
    }
}

//
```

```
static void algorithmChaos(int num){
    Random r = new Random();
    System.out.println("\nAlgorithm Chaos:");

    String[] alphaCap =
    {
        "A", "E", "I", "O", "U",
        "B", "C", "D", "F", "G", "H", "J", "K", "L", "M",
        "N", "Ng", "P", "Q", "R", "S", "T", "V", "W", "X", "Y", "Z",
    };

    String[] alpha =
    {
        "a", "e", "i", "o", "u",
        "b", "c", "d", "f", "g", "h", "j", "k", "l", "m",
        "n", "ng", "p", "q", "r", "s", "t", "v", "w", "x", "y", "z",
    };

    for(int j = 0; j < num; j++){

        int n = r.nextInt(10);
        if(n < 2){
            n = 2;
        }
        //System.out.println(n);

        for(int i = 0; i < n; i++){
            if(i ==0){
                System.out.print("\n" + alphaCap[r.nextInt(alphaCap.length)]);
            } else {
                System.out.print(alpha[r.nextInt(alpha.length)]);
            }
        }
    }
    System.out.println();
}
//

static void algorithm02(int num){
    Random r = new Random();
    System.out.println("\nAlgorithm 02:\n");

    //put em in a hierarchy
    for(int i = 0; i < num; i++){
        int numLetters = r.nextInt(10);
        while(numLetters < 2){
            numLetters = r.nextInt(10);
        }

        produceLetter(numLetters);
    }

}
public static void produceLetter(int numLetters){
    Random r = new Random();
    //categories

    String[] vowelDipthongs = {
        "a","e","i","o","u",
        "y", "w",
        "ae","ai","ao","au",
        "ea","ee","ei","eo","eu",
        "ia","ie","io","iu",
        "oa","oe","oi","oo","ou",
        "ua","ue","ui","uo",
        "aw","ew","iw","ow","uw",
        "wa","we","wi","wo","wu",
        "ay","ey","iy","oy","uy",
        "ya","ye","yi","yo","yu"
    };
    String[] liquids = {"l","r"};
    String[] nasalStops = {"m","n"};
    String[] sz = {
        "s","z",
        "es", "ez"
    };
    String[] africates = {"ch","j","č"};
    String h = ("h");
    String[] frictives = {"f","v", "th","sh","zh"};
    String[] bilabialOralStops = {"p","b"};
    String[] oralStops = {"t","d","k","g"};
    String ng = ("ng");

    //String[] plosives = {"p","b","t","d","k","g","ch","j"};
    //String[] continuants = {"l","r","m","n","ng"};
    String[] vowels = {"a","e","i","o","u"};
    //String[] glides = {"y","w"};
```

```
int cat = r.nextInt(10);
int x;

for(int i = 0; i < numLetters; i++){

    //if boundary
    if(cat == 0){
        x = r.nextInt(100);
        if(x < 80){
            System.out.print(vowels[r.nextInt(vowels.length)]);
        } else {
            System.out.print(vowelDipthongs[r.nextInt(vowelDipthongs.length)]);
        }
        while(cat == 0){
            cat = r.nextInt(10);
        }
    } else if(cat == 1){
        System.out.print(liquids[r.nextInt(liquids.length)]);
        cat = r.nextInt(10);
    } else if(cat == 2){
        System.out.print(nasalStops[r.nextInt(nasalStops.length)]);
        cat = r.nextInt(10);
    } else if(cat == 3){
        System.out.print(sz[r.nextInt(sz.length)]);
        cat = r.nextInt(10);
    } else if(cat == 4){
        System.out.print(africates[r.nextInt(africates.length)]);
        cat = r.nextInt(4);
    } else if(cat == 5){
        System.out.print(h);
        cat = r.nextInt(10);
        while(cat == 5){
            cat = r.nextInt(10);
        }
    } else if(cat == 6){
        System.out.print(frictives[r.nextInt(frictives.length)]);
        cat = r.nextInt(4);
    } else if(cat == 7){
        System.out.print(bilabialOralStops[r.nextInt(bilabialOralStops.length)]);
        cat = r.nextInt(4);
    } else if(cat == 8){
        System.out.print(oralStops[r.nextInt(oralStops.length)]);
        cat = r.nextInt(4);
```

```
            } else if(cat == 9){
                System.out.print(ng);
                cat = r.nextInt(4);
                cat = 0;
            }
            //if boundary
            x = r.nextInt(100);
            if(cat != 0){
                if(x < 50){ //chance for vowelDipthongs to always follow
                    cat = 0;
                }
            }

            //if boundary

        }
        System.out.println();
    }
    //




}

/*

     Basic Vowels

     a, e, i, o, u

     Basic Consonants

     b, c, ch, d, f, g, h, j, k, l, m,
     n, ng, p, q, r, s, sh, th, v, w, x, y, z, zh

     Orthography (Decorative Spelling)

     eau (o, beureau)

     Decorative Consonants

     [k] k, c, ck, ch, q, qh…
     [s] s, c, sc, Ps
```

[f] f, ph

ps ([s], as in psycho)(only for first syllables)
kn ([n], as in knot)
mn ([m], as in autumn)
gh ([f], rough; also, silent)

Silent decoratives

oj (o, as in Slavoj Zizek)
gh (as in, through; exclusive after ou, au,)
ps

Study on
Articulatory Phonetics

Nasals (m, n, exclude ng)
and
Liquids (l, r)
can always be followed by
Stops (p, b, t, d, k, g)
Frictives (f, v, th, s, z, sh, zh, exclude h)
Africates (ch, j)
.
The other way around will be awkward
and inelegant.
e.g. "apl", "evm","evn"
(that actually looks cool orthographically,
but phonetically, the speaker must supply
a vowel between the consonants.)
.
If followed the results might be
"elj", "arch", "enth","amp"
.
The nasal, [ng]
doesn't vibe well with the bilabial stops (p, b)
e.g. "angp"
this case, bilabial stops (p, b)
would have to be a standalone syllable
.
The nasal, [ng]
tho, vibes well with the rest of the
Stops, Frictives, and Africates
e.g. "beng-g", "bengk"

```
    .
    The Glides (w, y)
    preferably, must come before
    the Nasals, and Liquids
    e.g. "own", "awng", "ayn", "eym", "owl"

    I noticed that
    in the Nasals (l, r)
    [r] can precede [l] anytime (as in, "earl")
    but if [l] precedes [r] (as in, "calr")
    it's a no no.
    But in "Lrac" it's actually cool.


    a vowel can always precede or follow after
    itself, and all other consonants


    in the word "graft"
    there's seems a special relation to f and t
    that doesn't fit with frictive and stop category
    "bevt", for example.
    or, "befg".
    "ask" is smooth. hmm...
    maybe it's got to do with the voicedness.
    "asht", good enough.

*/
```