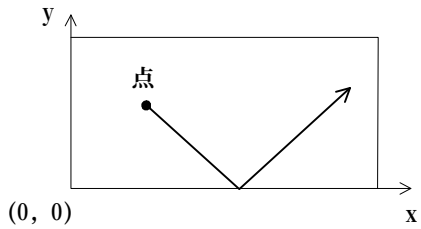
平成１７年度　春期　ＦＥ　午後問題　Java

問８　次の Java プログラムの説明及びプログラムを読んで，設問１，２に答えよ。

〔プログラムの説明〕

次の図のような長方形領域内を移動する点を描画するプログラムである。



点はクラス Point で表され，点の位置を表す座標（x, y）と速さ（speed）を保持する。速さは，正の値で単位時間に x 軸方向及び y 軸方向に移動する距離を表す。

図の長方形領域はクラス Space で与えられ，次のクラスメソッドを呼び出すことができる。

(1) public static int getMaxX()
　　長方形領域の x 座標の最大値（正の値）を返す。

(2) public static int getMaxY()
　　長方形領域の y 座標の最大値（正の値）を返す。

(3) public static void draw(Point)
　　引数で指定された点をその座標位置に描画する。

(4) public static void erase(Point)
　　引数で指定された点を消去する。

抽象クラス Motionは,点が移動する様子を示すために,コンストラクタに Point で与えられた点を初期値とし，点の描画，移動及び消去をこの順番で繰り返す。点の移動後の座標位置は抽象メソッド update で与えられる。

Motion のサブクラス SimpleMotion は，メソッド update と main を実装する。メソッド update は，長方形内では，一定の速さで直線運動し，長方形の境界で反射するような点の動きを表す。メソッド main はプログラムをテストする。

なお，メソッド main で生成される Point の座標の初期値は長方形領域内にあるものとし，点同士の衝突は考えないものとする。

〔プログラム１〕

```
public class Point {
   private int x, y, speed;
   public Point(int x, int y, int speed) {
      this.x = x; this.y = y;
      this.speed = speed;
   }
   public int getX() { return x; }
   public int getY() { return y; }
   public int getSpeed() { return speed; }
}
```

〔プログラム 2〕

```
public abstract class Motion implements
                                  Runnable {
   private Point point;

   public Motion(Point point) {
      this.point = point;
   }

   public void run() {
      while (true) {
         Space.draw(point);
         try {
            Thread.sleep(40);
         } catch (InterruptedException e) {}
         Point current = point;
                  a                ;
         Space.erase(current);
      }
   }

   public abstract Point update(Point point);
}
```

〔プログラム 3〕

```
public class SimpleMotion extends Motion {
   private int directionX = 1, directionY = 1;

   public SimpleMotion(Point point) {
      super(point);
   }

   public Point update(Point point) {
      int speed = point.getSpeed();
      int x = point.getX() + directionX * speed;
      int y = point.getY() + directionY * speed;
      if (x <= 0) {
         x = -x;
         directionX *= -1;
      }
      x %= 2 * Space.getMaxX();
      if (x >= Space.getMaxX()) {
         x = 2 * Space.getMaxX() - x;
         directionX *= -1;
      }
      if (y <= 0) {
         y = -y;
         directionY *= -1;
      }
      y %= 2 * Space.getMaxY();
      if (y >= Space.getMaxY()) {
         y = 2 * Space.getMaxY() - y;
         directionY *= -1;
      }
```

```
    return new Point(x, y, speed);
  }

  public static void main(String[] args) {
    Point[] points = {
      new Point(10,  20, 3),
      new Point(50,  10, 5),
      new Point(150, 60, 2)
    };
    for (int i = 0; i < points.length;
                                    i++) {
      new Thread(          b          ).
                                  start();
    }
  }
}
```

設問１　プログラム中の［　　］に入れる正しい答えを，解答群の中から選べ。

a に関する解答群

ア　current = update(current)

イ　current = update(point)

ウ　point = update(point)

エ　update(current)

オ　update(point)

b に関する解答群

ア　new Motion(points[i])

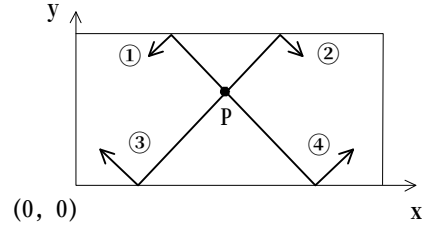
イ　new Point(points[i])

ウ　new Runnable(points[i])

エ　new SimpleMotion(points[i])

オ　points[i]

設問２　クラス SimpleMotion のコンストラクタに与えられた Point が次の図の 点 P で speed の値が１のとき，メソッド run を実行したときの点 P の動きとして正しい答えを，解答群の中から選べ。ただし，プログラム中の［　　］にはすべて正しい答えが入っているものとする。



解答群

ア　矢印 ① のように進む。

イ　矢印 ② のように進む。

ウ　矢印 ③ のように進む。

エ　矢印 ④ のように進む。

問 12　次の Java プログラムの説明及びプログラムを読んで，設問１，２に答えよ。

〔プログラムの説明〕

整数値の加減乗除の演算をする電卓プログラムである。入出力部分はテスト用のプログラムによって用意され，電卓本体部分のプログラムのテストができるようになっている。

(1) クラス CalculatorEvent は電卓のキーが押されたときに発生するイベントである。イベントにはタイプがありフィールド type の値で表される。タイプは DIGIT, OPERATOR, CLEAR のいずれかであり，それぞれ電卓の数字キー（０～９），演算キー（＋ など）及びイコールキー（=），クリアキー（C）を表す。タイプが DIGIT のときは数字キーに対応する数値を，OPERATOR のときは演算の種類を表す文字又は '=' をフィールド value に保持する。タイプが CLEAR のときは，value は使用しない。

(2) インタフェース CalculatorOutput は，電卓上に数値やエラーを表示するメソッド display を宣言する。

(3) クラス Calculator は電卓本体である。メソッド eventDispatched はイベントを受け取り，イベントのタイプに応じて演算処理などを行う。

なお，二つの数値に対する加減乗除の演算結果は Java の int 型の演算結果に一致するものとする。

(4) クラス CalculatorTest は Calculator をテストするプログラムである。CalculatorOutput を匿名クラスとして実装する。この実装では，メソッド display は指定された数値又は文字列を System.out に出力する。メソッド main は，引数 args[0] で与えられた文字列から CalculatorEvent を生成して，Calculator のメソッド eventDispatched を呼び出す。文字と電卓のキーの対応は，次の表のとおりである。

文字	電卓キー
'0' ～ '9'	数字キー（0 ～ 9）
'+'	加算キー（＋）
'-'	減算キー（－）
'*'	乗算キー（×）
'/'	除算キー（÷）
'='	イコールキー（＝）
'C'	クリアキー（C）

例えば，文字列“2＋7=”は，電卓キー 2，＋，7，＝が順に押されたことを表し，この文字列が引数 args[0] としてメソッド main に渡されたとき，プログラムは次のとおり出力する。

```
2
2
7
9
```

〔プログラム1〕

```
public class CalculatorEvent {
   public static final int DIGIT = 1;
   public static final int OPERATOR = 2;
   public static final int CLEAR = 3;

   private int type, value;

   public CalculatorEvent(int type) {
       [ a ];
   }
   public CalculatorEvent(int type,
                          int value) {
      if (type < DIGIT || type > CLEAR)
         throw new IllegalArgumentException();
      this.type = type; this.value = value;
   }
   public int getType() { return type; }
   public int getValue() { return value; }
}
```

〔プログラム2〕

```
public interface CalculatorOutput {
   public void display(int value);
   public void display(String value);
}
```

〔プログラム3〕

```
public class Calculator {
   private int accumulator = 0, register = 0;
   private int operator = 0;
   private CalculatorOutput output;

   public Calculator(CalculatorOutput output) {
      this.output = output;
   }

   public void event Dispatched
                  (CalculatorEvent event) {
      switch (event.getType()) {
      case CalculatorEvent.DIGIT:
         if (operator == '=') {
            register = 0; operator = 0;
         }
         register = register * 10 + event.
                                  getValue();
         output.display(register);
         break;
      case CalculatorEvent.OPERATOR:
         try {
            register = calculate();
            output.display(register);
            accumulator = register;
            operator = event.getValue();
         } catch (ArithmeticException e) {
            output.display("Error");
            accumulator = 0; operator = 0;
         }
         if (operator != '=')
            register = 0;
         break;
      case CalculatorEvent.CLEAR:
         register = 0;
         accumulator = 0;
         operator = 0;
         output.display(register);
         break;
      }
   }

   private int calculate() {
      switch (operator) {
      case '+':
         return accumulator + register;
      case '-':
         return accumulator - register;
      case '*':
         return accumulator * register;
      case '/':
         return accumulator / register;
      }
      return register;
   }
}
```

〔プログラム4〕

```
public class CalculatorTest {
   public static void main(String[] args) {
      Calculator calc = new Calculator(
        [ b ] {
          public void display(int value) {
             System.out.println(value);
          }
          public void display(String value) {
             System.out.println(value);
          }
        });
      String keys = args[0];
      for (int i = 0; i < keys.length(); i++) {
         char c = keys.charAt(i);
         CalculatorEvent event = null;
         if (c >= '0' && c <= '9') {
            event = new CalculatorEvent(
```

```
                    c                    );
        } else if (c == '=' || c == '+' ||
                   c == '-' || c == '*' ||
                   c == '/') {
           event = new CalculatorEvent(
               CalculatorEvent.OPERATOR, c);
        } else if (c == 'C') {
           event = new CalculatorEvent(
                    CalculatorEvent.CLEAR);
        }
        if (event != null)
           calc.eventDispatched(event);
     }
   }
}
```

設問１　プログラム中の [　　] に入れる正しい答えを，解答群の中から選べ。

a に関する解答群

ア　CalculatorEvent(type, 0)

イ　new CalculatorEvent(type, 0)

ウ　return new CalculatorEvent(type, 0)

エ　super(type, 0)

オ　this(type, 0)

b に関する解答群

ア　implements CalculatorOutput()

イ　interface CalculatorOutput()

ウ　new CalculatorOutput()

エ　new Temp() implements CalculatorOutput

オ　public class Temp implements CalculatorOutput

c に関する解答群

ア　c - '0', CalculatorEvent.DIGIT

イ　c, CalculatorEvent.DIGIT

ウ　CalculatorEvent.DIGIT

エ　CalculatorEvent.DIGIT, c

オ　CalculatorEvent.DIGIT, c - '0'

設問２　次の表は文字列を引数としてメソッド main を実行したときに，最後に出力された結果を表している。表中の [　　] に入れる正しい答えを，解答群の中から選べ。ただし，プログラム中の [　　] にはすべて正しい答えが入っているものとする。

文字列	出力
3+4*5=	35
3*4***=	[　d　]
3*4=+5	[　e　]
3+4/0=	[　f　]

解答群

ア　0　　イ　3　　ウ　4　　エ　5

オ　7　　カ　12　　キ　17　　ク　53

ケ　/ by zero　　コ　Error

平成１７年度　秋期　ＦＥ　午後問題　Java

問８　次の Java プログラムの説明及びプログラムを読んで，設問に答えよ。

〔プログラムの説明〕

英文のテキストを処理し，単語の出現回数を数えるプログラムの一部である。テキストは，単語，空白，コンマ及びピリオドで構成され，単語はアルファベットだけで構成される。単語中の大文字はすべて小文字に変換して取り扱う。さらに，単語の出現回数だけでなく，アルファベットの各文字が単語の先頭に現れた回数を数えられるようにする。

(1)　WordTable は，与えられた文字列から単語を切り出し，出現回数を数えるクラスである。実際に単語を数える処理は，コンストラクタで指定する別のオブジェクトで行う。

(2)　Counter は，単語を数える処理のインタフェースである。

(3)　WordCounter は，単語の出現回数を数えるクラスである。

(4)　FirstLetterCounter は，アルファベットの各文字が単語の先頭に現れた回数を数えるクラスである。

(5)　Test は，テスト用のメインプログラムである。実行例を図に示す。

```
wordCountTable:
java(2)
hello(1)
wonderful(1)
world(1)
is(1)

firstLetterCountTable:
h(1)
i(1)
j(2)
w(2)
```

図　クラス Test の実行例

java.util.StringTokenizer は，指定された区切り文字で，文字列を字句単位（トークン）に分解するクラスである。次のメソッドをもつ。

public boolean hasMoreTokens()
　文字列に利用できるトークンがまだあるかどうかを判定し，結果を返す。
public String nextToken()
　次のトークンを返す。

java.util.Map は，キーと値を関連付けて管理するインタフェースであり， java.util.HashMap は，そのインタフェースを実装したクラスである。キーを指定して，そのキーに値を関連付けたり，そのキーに関連付けられた値を取り出したりすることができる。次のメソッドをもつ。

public Object get(Object key)
　key に関連付けられた値を返す。
public Object put(Object key, Object value)
　key に value を関連付ける。
public boolean containsKey(Object key)
　key に関連付けられた値がある場合に true を返す。
public Set keySet()
　マップに含まれるキーの集合を返す。

java.util.Iterator は，要素を順番に取り出すための操作を提供するインタフェースである。次のメソッドをもつ。

public boolean hasNext()
　次の要素がある場合に true を返す。
public Object next()
　次の要素を返す。

〔プログラム１〕

```
import java.util.StringTokenizer;

public class WordTable {
  private [　　a　　] counter;
  public WordTable([　　a　　]
                                counter){
    this.counter = counter;
  }
  public void put(String line) {
    StringTokenizer st =
         new StringTokenizer(line, " ,.");
    while (st.hasMoreTokens())
          counter.put(st.nextToken().
                          toLowerCase());
  }
  public String toString() {
    return counter.toString();
  }
}
```

〔プログラム 2〕

```
public interface Counter {
  void put(String str);
}
```

〔プログラム 3〕

```
import java.util.Map;
import java.util.HashMap;
import java.util.Iterator;

public class WordCounter [　　b　　] {
  private Map freq = new HashMap();
  public void put(String str) {
    int count = [　　c　　];
    if (freq.containsKey(str))
      count += ((Integer) freq.get(str)).
                                intValue();
    freq.put(str, new Integer(count));
  }
  public String toString() {
    StringBuffer buf = new StringBuffer();
    for (Iterator it = freq.keySet().
                                iterator();
         it.hasNext(); ) {
      String word = (String) it.next();
      buf.append(word + "(" + freq.get(word)
                                  + ")\n");
    }
    return buf.toString();
  }
}
```

〔プログラム 4〕

```
public class FirstLetterCounter
                    [　　b　　] {
  private int[] flFreq = new int[26];
  public void put(String str) {
    [　　d　　]++;
  }
  public String toString() {
    StringBuffer buf = new StringBuffer();
    for (int i = 0; i < flFreq.length; i++) {
      if (flFreq[i] != 0)
        buf.append((char) ('a' + i)
                  + "(" + flFreq[i] + ")\n");
    }
    return buf.toString();
  }
}
```

〔プログラム 5〕

```
public class Test {
  public static void main(String[] args) {
    String text = "Hello java world. Java is
```

```
                                    wonderful.";
        WordTable wordCountTable =
            new WordTable(new WordCounter());
        WordTable firstLetterCountTable =
            new WordTable
                    (new FirstLetterCounter());
        wordCountTable.put(text);
        firstLetterCountTable.put(text);
        System.out.println("wordCountTable:\n"
                           + wordCountTable);
        System.out.println
               ("firstLetterCountTable:\n" +
                      firstLetterCountTable);
    }
}
```

設問　プログラム中の［　　］に入れる正しい答えを，解答群の中から選べ。

a に関する解答群

ア　Counter　　イ　FirstLetterCounter

ウ　Object　　エ　WordCounter

オ　WordTable

b に関する解答群

ア　extends Counter

イ　extends Object

ウ　extends WordTable

エ　implements Counter

オ　implements Object

カ　implements WordTable

c に関する解答群

ア　-1

イ　0

ウ　1

エ　((Integer) freq.get(str)).intValue()

オ　str.length()

d に関する解答群

ア　flFreq['a' + str.charAt(0)]

イ　flFreq['a' + str.charAt(1)]

ウ　flFreq[str.charAt(0)]

エ　flFreq[str.charAt(1)]

オ　flFreq[str.charAt(0) - 'a']

カ　flFreq[str.charAt(1) - 'a']

問 12　次の Java プログラムの説明及びプログラムを読んで，設問１，２に答えよ。

〔プログラムの説明〕

A を起点，D を終点とする四つの駅 A，B，C，D からなる路線の各駅に設置された自動改札機の処理を模したプログラムである。この路線では片道切符（以下，切符という）とプリペイドカード（以下，カードという）の２種類の乗車券が使用できる。

路線の運賃は駅間の距離で決められる。距離が 4 km までは 120 円（この運賃を初乗り運賃という）で，それを超えると 2 km ごとに 30 円加算される。2 km 未満は 2 km に切り上げる。例えば，距離が 7 km のとき，運賃は 180 円である。

切符には，乗車駅で自動改札機を通って入場するとき，乗車駅の情報が記録される。降車駅で自動改札機を通って出場するとき，運賃が計算され，金額が不足しているときはゲートが閉じられ，出場できない。一度使用した切符は無効となる。この路線では切符を発券した駅にかかわらず，どの駅の自動改札機からでも入場し，乗車できる。例えば，A 駅で発券された切符で B 駅の自動改札機から入場できる。

カードには，乗車駅で自動改札機を通って入場するとき，乗車駅の情報が記録される。このとき，カードの残高が 0 円の場合は，ゲートが閉じられ，入場できない。降車駅で自動改札機を通って出場するとき，精算処理が行われる。すなわち，運賃が計算され，カードの残高から引かれる。このとき，カードの残高が運賃に満たない場合は，ゲートが閉じられ，出場できない。

クラス Line は路線を表す。メソッド getFare は与えられた距離から運賃を計算して返す。

クラス Gate は，自動改札機を表す。クラス Line のフィールド A，B，C，D は Gate のインスタンスであり，それぞれ A，B，C，D の各駅に設置された自動改札機を表す。コンストラクタ及び各メソッドは，次の処理を行う。

(1)　コンストラクタは，Gate のインスタンスを生成する。最初の引数に駅名，２番目の引数に路線の起点である A 駅からの距離を指定する。

(2)　メソッド enter は，自動改札機を通って入場するときの処理を行う。乗車券が適正でない場合はゲートを閉じる。入場処理が正常に行われた場合は，乗車券に乗車駅の情報を記録する。

(3)　メソッド exit は，自動改札機を通って出場するときの処理を行う。乗車券の金額(残高)が不足するなど，適正でない場合はゲートを閉じる。

(4)　メソッド open 及び close は，それぞれゲートの開閉を表すメッセージを出力する。

抽象クラス Ticket は，この路線の乗車券を表し，このクラスを継承して切符やカードを定義する。コンストラクタ及び各メソッドは，次の処理を行う。

(1)　コンストラクタは，購入時の金額を初期値として乗車券に保持する。

(2)　メソッド getValue は，呼び出された時点での乗車

券の金額（残高）を返す。

(3) メソッド adjustValue は，必要であれば精算処理を行う。

(4) メソッド deduct は，引数で指定された金額を乗車券の金額（残高）から差し引いて金額（残高）を更新する。

(5) メソッド setOrigin は，指定された Gate を乗車駅として記録する。 null が指定されたときは，乗車駅の記録を消去する。

(6) メソッド getOrigin は，記録されている乗車駅を返す。記録されていないときは null を返す。

クラス OneWayTicket は切符を表し，クラス PrepaidCard はカードを表す。それぞれの乗車券の処理で，抽象メソッドを実装し，必要に応じて Ticket のメソッドをオーバライドする。

〔プログラム１〕

```
public final class Line {
  public static final Gate A
                       = new Gate("A", 0);
  public static final Gate B
                       = new Gate("B", 5);
  public static final Gate C
                       = new Gate("C", 8);
  public static final Gate D
                       = new Gate("D", 14);

  public static int getFare(int distance) {
    return 120 + (Math.max(distance - 3, 0)
                                   / 2) * 30;
  }
}
```

〔プログラム２〕

```
public class Gate {
  private final String name;
  private final int distance;

  public Gate(String name, int distance) {
    this.name = name;
    this.distance = distance;
  }

  public void enter(Ticket ticket) {
    if (ticket.isValid() &&
              ticket.getOrigin() == null) {
      [          a          ] ;
      open();
    } else {
      close();
    }
  }

  public void exit(Ticket ticket) {
    Gate origin = ticket.getOrigin();
    if (origin != null) {
      int d = Math.abs(origin.distance -
                                  distance);
      int fare = Line.getFare(d);
      if ( [          b          ] ) {
        ticket.adjustValue(fare);
        ticket.setOrigin(null);
        open();
        return;
      }
    }
    close();
  }

  private void open() { System.out.println
                        (name + ": open"); }
  private void close() {
    System.out.println(name + ": closed");
  }
}
```

〔プログラム３〕

```
public abstract class Ticket {
  private Gate origin;
  private int value;

  public Ticket(int value) {
    this.value = value;
  }

  public int getValue() { return value; }

  public void deduct(int amount) {
                          value -= amount; }

  public void setOrigin(Gate gate) {
                            origin = gate; }

  public Gate getOrigin() { return origin; }

  public abstract void adjustValue
                               (int amount);

  public abstract boolean isValid();
}
```

〔プログラム４〕

```
public class OneWayTicket extends Ticket {
  private boolean valid = true;

  public OneWayTicket(int value) {
    [          c          ];
  }

  public void setOrigin(Gate gate) {
```

```
      super.setOrigin(gate);
      if (gate == null)
         valid = false;
   }

   public void adjustValue(int amount) { }

   public boolean isValid() { return valid; }
}
```

〔プログラム５〕

```
public class PrepaidCard extends Ticket {
   public PrepaidCard(int value) {
      [        c        ];
   }

   public void adjustValue(int amount)
                       { deduct(amount); }

   public boolean isValid() {
      return getValue() > 0;
   }
}
```

設問１　プログラム中の [　　] に入れる正しい答えを，解答群の中から選べ。

a に関する解答群

ア　ticket.setOrigin(Line.A)

イ　ticket.setOrigin(Line.D)

ウ　ticket.setOrigin(null)

エ　ticket.setOrigin(this)

オ　ticket.setOrigin(ticket)

b に関する解答群

ア　ticket.getValue() < fare

イ　ticket.getValue() <= fare

ウ　ticket.getValue() == fare

エ　ticket.getValue() > fare

オ　ticket.getValue() >= fare

c に関する解答群

ア　super()

イ　super(this)

ウ　super(value)

エ　super(); this.value = value

オ　this(value)

設問２　この路線で，新しいタイプの乗車券を発売することになった。この乗車券は発券した時刻から 24 時間以内は全駅で乗り降り自由な乗車券である。発券から 24 時間経過すると出場はできるが入場はできなくなる。これに伴い，抽象クラス Ticket を継承して新しいクラスを定義し，クラス Gate には修正を加えずにこの乗車券をサポートしたい。このクラスのコンストラクタ及びメソッドの処理を次の表にまとめた。表中の [　　] に入れる正しい答えを，解答群の中から選べ。ここで，解答群中の value は，クラス Ticket のフィールド value であり，コンストラクタで初期値を設定し，値はメソッド getValue で得るものとする。

コンストラクタ及びメソッド	処理
コンストラクタ	[　d　]
メソッド getValue	[　e　]
メソッド setOrigin	スーパクラスで定義されたとおり
メソッド getOrigin	スーパクラスで定義されたとおり
メソッド adjustValue	何もしない（メソッド本体が文を含まない）。
メソッド isValid	[　f　]

解答群

ア　value が路線の最高運賃（全運賃の最大値）以上のときだけ true を返す。それ以外は false を返す。

イ　value が路線の初乗り運賃以上のときだけ true を返す。それ以外は false を返す。

ウ　value の初期値に，理論的に 24 時間かかっても使い切れない金額を設定する。

エ　value の初期値に，路線の最高運賃（全運賃の最大値）を設定し，発券時刻を記録する。

オ　スーパクラスで定義されたとおり

カ　常に 0 を返す。

キ　常に路線の初乗り運賃の値を返す。

ク　何もしない（メソッド本体が文を含まない）。

ケ　メソッド deduct を，amount を引数として呼び出す。

コ　呼び出されたときの時刻がインスタンスに格納されている発券時刻から 24 時間以内のときだけ true を返す。それ以外は false を返す。

平成１７年度　春期　ＦＥ　午後解答　Java

問８

設問１
a－ウ　　b－エ

設問２
イ

問 12

設問１
a－オ　　b－ウ　　c－オ

設問２
d－ア　　e－エ　　f－コ

平成１７年度　秋期　ＦＥ　午後解答　Java

問８

a－ア　　b－エ　　c－ウ　　d－オ

問 12

設問１
a－エ　　b－オ　　c－ウ

設問２
d－エ　　e－オ　　f－コ