問11　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。
（Javaプログラムで使用するAPIの説明は，この冊子の末尾を参照してください。）

〔プログラムの説明〕

整数値の加減乗除の演算をする電卓のプログラムである。この電卓は，数字キー，加減乗除の各演算キー，イコールキー及びクリアキーをもつ。プログラムは，キーが押されたとき，それぞれのキーに対応する処理を実行する。数値などの表示は，System.out.println を呼び出して行う。

(1) インタフェース Key は，電卓のキーが押されたときの処理を実行するメソッドを定義する。

メソッド operateOn は，引数で与えられたクラス java.util.Stack のインスタンス（以下，スタックという）に対して，キーに対応する処理を実行する。

(2) 列挙 DigitKey は，数字キーを表す定数 DIGIT0 ～ DIGIT9 を定義する。

メソッド operateOn は，キーを 10 進数の入力として処理する。引数で与えられたスタックの先頭に格納されている値は 0（初期値）又は入力中の数値であり，その値を更新する。

(3) 列挙 OperationKey は，加減乗除の各演算キー，イコールキー及びクリアキーを表す定数 ADD，SUBTRACT，MULTIPLY，DIVIDE，EQUAL 及び CLEAR を定義する。

メソッド operateOn は，加減乗除の各演算キーに対応する演算を，スタックの内容に対して実行する。

(4) クラス Calculator は，電卓本体を表す。フィールド stack は，電卓内部の数値の状態を表すスタックを保持する。フィールド pendingKey は，演算に必要な数値の入力が終わるまで演算キーを保持する。また，イコールキーが押されたときは，イコールキーを保持する。例えば，キーの定数 DIGIT2，ADD，DIGIT4 が順に処理されたとき，スタックに格納されている値は先頭から 4，2 であり，pendingKey の値は ADD である。次にキーの定数 EQUAL が処理されたとき，演算キーADD の加算処理が実行され，スタックに格納されている値は 6 となり，pendingKey の値は EQUAL となる。ここで，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。

メソッド onKeyPressed は，電卓のキーが押されたときに呼び出される。押されたキーは，引数で与えられる。押されたキー及び電卓の内部状態に基づいて，処理を実行する。

(5) クラス CalculatorTest は，クラス Calculator をテストするプログラムである。

メソッド main は，まず，文字と電卓の各キーとの対応を作成し，クラス Calculator のインスタンスを生成する。次に，引数で与えられた文字列の各文字をキーの定数に変換し，そのキーの定数を引数としてクラス Calculator のインスタンスのメソッド onKeyPressed を呼び出す。例えば，メソッド main の引数として文字列“2*3=”が与えられたとき，それぞれの文字を，キーの定数 DIGIT2, MULTIPLY, DIGIT3, EQUAL に変換し，逐次それぞれのキーの定数を引数としてメソッド onKeyPressed を呼び出す。メソッド main を実行したときの出力を図1に示す。

```
DIGIT2
2
MULTIPLY
2
DIGIT3
3
EQUAL
6
```

図1　メソッド main を実行したときの出力

〔プログラム1〕

```
import java.util.Stack;

public interface Key {
   public void operateOn(Stack<Integer> stack);
}
```

〔プログラム2〕

```
import java.util.Stack;

enum DigitKey [ a ] Key {
   DIGIT0, DIGIT1, DIGIT2, DIGIT3, DIGIT4,
   DIGIT5, DIGIT6, DIGIT7, DIGIT8, DIGIT9;
```

```
    public void operateOn(Stack<Integer> stack) {
        stack.push( [ b ] * 10 + [ c ] );
    }
}
```

〔プログラム3〕

```
import java.util.Stack;

enum OperationKey [ a ] Key {
    ADD, SUBTRACT, MULTIPLY, DIVIDE, EQUAL, CLEAR;

    public void operateOn(Stack<Integer> stack) {
        if (this == EQUAL || this == CLEAR) {
            return;
        }
        int val2 = stack.pop();
        int val1 = stack.pop();
        stack.push(calculate(val1, val2));
    }

    private int calculate(int val1, int val2) {
        switch ( [ d ] ) {
        case ADD:
            return val1 + val2;
        case SUBTRACT:
            return val1 - val2;
        case MULTIPLY:
            return val1 * val2;
        case DIVIDE:
            return val1 / val2;
        default:
            throw new AssertionError(toString());
        }
    }
}
```

〔プログラム4〕

```
import java.util.Stack;

public class Calculator {
    private final Stack<Integer> stack = new Stack<Integer>();
```

```
    private Key pendingKey;

    public Calculator() {
        stack.push(0);
    }

    public void onKeyPressed(Key key) {
        System.out.println(key);
        if (key instanceof DigitKey) {
            if (pendingKey == OperationKey.EQUAL) {
                reset();
            }
            key.operateOn(stack);
            System.out.println(stack.peek());
        } else if (key == OperationKey.CLEAR) {
            reset();
            System.out.println(stack.peek());
        } else {
            try {
                if (pendingKey != null) {
                    pendingKey.operateOn(stack);
                }
                System.out.println(stack.peek());
                pendingKey = key;
                if (key != OperationKey.EQUAL) {
                    stack.push(0);
                }
            } catch (ArithmeticException e) {
                System.out.println("Error");
                reset();
            }
        }
    }

    private void reset() {
        stack.clear();
        stack.push(0);
        pendingKey = null;
    }
}
```

〔プログラム5〕

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

public class CalculatorTest {
   public static void main(String[] args) {
      Map<Character, [ e ]> map = new HashMap<Character, [ e ]>();
      // 文字と列挙OperationKeyの定数の対応をmapに格納する。
      for (OperationKey key : OperationKey.values())
         map.put("+-*/=C".charAt(key.ordinal()), key);
      // 数字と列挙DigitKeyの定数の対応をmapに格納する。
      for (DigitKey key : DigitKey.values())
         map.put("0123456789".charAt(key.ordinal()), key);

      Calculator calc = new Calculator();
      String chars = args[0];
      // charsの各文字をキーの定数に変換し，メソッドonKeyPressedを呼び出す。
      for (int i = 0; i < chars.length(); i++) {
         calc.onKeyPressed(map.get(chars.charAt(i)));
      }
   }
}
```

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　extends	イ　implements	ウ　imports
エ　inherits	オ　requires	カ　throws

b，c に関する解答群

ア　ordinal()	イ　stack.peek()	ウ　stack.pop()
エ　stack.push(0)	オ　stack.push(ordinal())	カ　values()

d に関する解答群

ア　DigitKey	イ　Key	ウ　stack.pop()
エ　this	オ　val1	カ　val2

e に関する解答群

ア Calculator	イ Character	ウ DigitKey
エ Integer	オ Key	カ OperationKey

設問2　表1は，文字列を引数としてメソッド main を実行したときの出力の最後の行（図1の場合は6）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。ここで，プログラム中の［　　］には，全て正しい答えが入っているものとする。

表1　文字列（引数）と出力（最後の行）

文字列（引数）	出力（最後の行）
2*6/3=	4
-2=	-2
2*4==	8
2*4C2=	f
8/2/=	g

f，g に関する解答群

ア 0	イ 2	ウ 4
エ 8	オ 16	カ 32
キ 64	ク ArithmeticException	ケ Error