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〕

D 君は，社内向けの応用プログラム（以下，アプリケーションという）で共通に使うライブラリを開発するチームに属している。ライブラリの次のバージョンで，時間に関するクラスを幾つか用意することになり，D 君は期間（時間間隔）を表すクラス Period の開発を担当することになった。このクラスは，“有効期間は，今週月曜日の午前 0 時から金曜日の終わりまで（土曜日の午前 0 時以降は，無効）”など，ある日付及び時刻（以下，日時という）から別の日時までの時間間隔を表すことを想定している。期間の基準になる日時を始点，終わりを表す日時を終点という。また，ある始点から過去に遡って期間を表すことも想定している。D 君が定めたクラス Period の外部仕様は，次のとおりである。

クラス Period は，始点から終点までの期間を表す。期間には，始点は含まれるが終点は含まれない。始点と終点が同じ日時の場合は，期間が空であると定義し，いかなる日時も含まないものとする。

(1) コンストラクタは，引数で与えられた始点（start）と終点（end）から期間を表すインスタンスを生成する。日時は，クラス Date のインスタンスで与えられ，協定世界時 1970 年 1 月 1 日午前 0 時（以下，この日時をエポックという）以降でなければならない。引数 start 又は end が null の場合は，NullPointerException を，日時がエポックよりも前の場合は，IllegalArgumentException を投げる。

(2) メソッド getStart：始点を返す。

(3) メソッド getEnd：終点を返す。

(4) メソッド getLength：期間の長さ（ミリ秒）を返す。終点が始点よりも前の場合は，負の値で返す。期間が空の場合は，0 を返す。

(5) メソッド isBackward：終点が始点よりも前の場合は，true を返す。それ以外は，false を返す。期間が空の場合は，false を返す。

(6) メソッド contains：引数で与えられた日時が期間に含まれる場合は，true を返す。それ以外は，false を返す。期間が空の場合は，false を返す。引数が

null の場合は，NullPointerException を投げる。

この仕様でチームの承認を得て，次のプログラム 1 を作成した。

〔プログラム 1〕

```
import java.util.Date;

public final class Period {
   private static final Date EPOCH = new Date(0L);
   private final Date start, end;

   public Period(Date start, Date end) {
      if (start == null || end == null) {
         throw new NullPointerException();
      }
      if (start.compareTo(EPOCH) < 0
          || end.compareTo(EPOCH) < 0) {
         throw new IllegalArgumentException();
      }
      this.start = start;
      this.end = end;
   }

   public Date getStart() { return start; }

   public Date getEnd() { return end; }

   public long getLength() {
      return end.getTime() - start.getTime();
   }

   public boolean isBackward() {
      return end.compareTo(start) < 0;
   }

   public boolean contains(Date time) {
      return (time.compareTo(start) >= 0
              && time.compareTo(end) < 0)
          || (time.compareTo(start) [ a ] 0
              && time.compareTo(end) [ b ] 0);
   }
}
```

次に，D 君は，テスト用にプログラム 2 を作成した。問題が検出されなければ，プログラム 2 は何も出力せずに終了する。問題を検出した場合は，RuntimeException を投げる。

〔プログラム2〕

```
import java.util.Date;

public class PeriodTest {
   private static final long DELTA = 60 * 60 * 1000L;

   // コンストラクタが，正しくない引数に対して仕様どおりに例外を投げること
   // を確認する。引数 expected は，期待される例外の型をクラスで指定する。
   private static void testException(Date start, Date end,
               Class<? extends RuntimeException> expected) {
      try {
         Period period = new Period(start, end);
      } catch (RuntimeException e) {
         if (e.getClass() == expected) {
            return;
         }
         throw new RuntimeException("unexpected exception", e);
      }
      throw new RuntimeException("no " + expected + " thrown");
   }

   // 期間の長さ及び方向（始点と終点の前後関係）の整合性を確認する。
   private static void testConsistency(Period period,
                                       long length) {
      if (period.getLength() != length) {
         throw new RuntimeException("invalid getLength() value");
      }
      if (period.isBackward() != (length [ c ] 0)) {
         throw new RuntimeException("isBackward failed");
      }
   }

   // メソッド contains が，期間に含まれるデータ及び含まれないデータ
   // に対して正しく判定することを確認する。また，メソッドの引数が
   // null のとき，NullPointerException を投げることを確認する。
   private static void testContains(Period period, long[] valid,
                                    long[] invalid) {
      for (long time : valid) {
         if (!period.contains(new Date(time))) {
            throw new RuntimeException("failed with valid: " + time);
         }
      }
      for (long time : invalid) {
         if (period.contains(new Date(time))) {
            throw new RuntimeException("failed with invalid: "
                                       + time);
         }
      }
      try {
         period.contains(null);
```

```
            throw new RuntimeException("no NPE thrown");
        } catch (NullPointerException e) {
        }
    }

    public static void main(String[] args) {
        testException(null, new Date(0L), [ d ]);
        testException(new Date(0L), null, [ d ]);
        testException(new Date(-1L), new Date(0L), [ e ]);
        testException(new Date(0L), new Date(-1L), [ e ]);

        long now = System.currentTimeMillis();
        Date start = new Date(now);
        Date end = new Date(now + DELTA);
        final Period period = new Period(start, end);
        testConsistency(period, DELTA);
        testContains(period,
            new long[] { now, now + 1, now + DELTA - 1 },
            new long[] { now - 1, now + DELTA, now + DELTA + 1 });

        Date backwardEnd = new Date(now - DELTA);
        final Period backwardPeriod = new Period(start, backwardEnd);
        testConsistency(backwardPeriod, -DELTA);
        testContains(backwardPeriod,
            new long[] { now, now - 1, now - DELTA + 1 },
            new long[] { now + 1, now - DELTA, now - DELTA - 1 });

        final Period nullPeriod = new Period(start, start);
        testConsistency(nullPeriod, 0);
        testContains(nullPeriod, new long[0],
                     new long[] { now - 1, now, now + 1 });
                                                          ←―― α
    }
}
```

設問1　プログラム 1 及び 2 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2を実行したとき，例外は発生せず正常に終了するものとする。

a～cに関する解答群

ア　!=	イ　<	ウ　<=
エ　==	オ　>	カ　>=

d，eに関する解答群

ア　IllegalArgumentException

イ　IllegalArgumentException.class

ウ　IllegalArgumentException.getClass()

エ　new IllegalArgumentException()

オ　new NullPointerException()

カ　NullPointerException

キ　NullPointerException.class

ク　NullPointerException.getClass()

設問2　次の記述中の［　　］に入れる正しい答えを，解答群の中から選べ。

チーム内でのコードレビューも終わり，クラス Period を含むライブラリが評価版として社内にリリースされた。それからしばらくして，アプリケーション開発チームから，アプリケーションの実行中に Period のインスタンスが表す期間が変わってしまうという指摘を受けた。D君は，プログラムを見直したがクラスもフィールドも最終（final）なので，生成後にインスタンスの状態が変化することはないと考えた。そこで，D君は，アプリケーション開発チームと一緒にデバッグを行った。

その結果，アプリケーションでは，Period のメソッド getStart で始点を取得し，その Date インスタンスの表す日時を変更して別の Period のインスタンスを生成するときの引数に使用していることが分かった。これが原因で，先に生成済みの Period のインスタンスの始点が変更されていた。すなわち，Period インスタンス自体は不変でも，それから参照される Date インスタンスは可変なので，表している期間が変わってしまうことがある。したがって，Period のインスタンスで始点及び終点を保持する場合は，外部からの変更ができないようにする必要がある。また，クラス Date は，最終（final）ではないので，どのような下位クラスでも作成可能であり，上書きしたメソッドについて挙動の変更が可能である。これらは，データの改ざんを許すことになるので，セキュリティ上の問題でもある。これらの問題を修正するために，D君は，クラス Period に次

の変更を加えた。

(1) コンストラクタの処理において，this.start の代入文の右辺を new Date(start.getTime())に変更した。this.end の代入文についても同様の変更をした。この変更によって，フィールド start 及び end は，それぞれ引数と同じ値をもつ別の java.util.Date のインスタンスを参照する。

(2) メソッド getStart の返却値を［ f ］に変更した。メソッド getEnd についても同様の変更をした。

D君は，プログラム2のメソッド main の α で示した部分に次のプログラムを追加し，正しく修正されたことを確認した。

```
start.setTime(now - DELTA);
end.setTime(now + DELTA * 2);
testConsistency(period, [ g ]);
testContains(period,
   new long[] { now, now + 1, now + DELTA - 1 },
   new long[] { now - 1, now + DELTA, now + DELTA + 1 });
Date newStart = period.getStart();
newStart.setTime(newStart.getTime() - DELTA);
Date newEnd = period.getEnd();
newEnd.setTime(newEnd.getTime() + DELTA);
testConsistency(period, [ g ]);
testContains(period,
   new long[] { now, now + 1, now + DELTA - 1 },
   new long[] { now - 1, now + DELTA, now + DELTA + 1 });
```

fに関する解答群

ア (Date) start.clone()　　イ null

ウ start.clone()　　エ start.getTime()

オ start.setTime(start.getTime())

gに関する解答群

ア 0　　イ DELTA　　ウ DELTA * 2

エ DELTA * 3　　オ DELTA / 2　　カ DELTA / 3