錬成問題

▪ クラスを構成する主な要素は、下図に示す［　(1)　］、［　(2)　］、［　(3)　］である。

```
//--- 会員クラス ---//
class Member {
   private String name;     // 名前
   private int no;          // 会員番号
   private int age;         // 年齢

   Member(String name, int no, int age) {
      this.name = name;
      this.no = no;
      this.age = age;
   }

   // 会員番号を取得
   int getNo() {
      return no;
   }

   // 会員番号を変更
   void setNo(int no) {
      this.no = no;
   }

   // 名前・会員番号・年齢を表示
   void print() {
      System.out.println("No." + no + "：" + name +
                         " (" + age + "歳) ");
   }
}
```

(1)　(2)　(3)

▪ クラス *Member* 型のクラス型変数 *takaoka* の宣言は、以下のように行う。

```
Member ［　(4)　］;
```

▪ クラス *Member* 型の実体を生成して、名前（*name*）・番号（*no*）・年齢（*age*）を、それぞれ"高岡"，75，55 で初期化する式は、次のようになる。

```
［　(5)　］
```

▪ 前問と同じようにインスタンスを生成するとともに、そのインスタンスを、前々問で宣言されたクラス型変数 *takaoka* が参照するように設定するには、以下のように代入を行えばよい。

```
［　(6)　］ = ［　(7)　］;
```

▪ クラス *Member* 型の実体を生成して、名前・番号・年齢を、それぞれ "南郷"，35，65 で初期化するとともに、そのインスタンスを参照するクラス型変数 *nango* を宣言するには、以下のように宣言する。

```
Member  (8)  =  (9) ;
```

▪ 以下に示すのは、*nango* が参照するインスタンスの会員番号を 80 に変更し、それから変更された会員番号を表示するプログラムである。

```
 (10) ;                                          // 会員番号を80に変更
System.out.println("会員番号：" +  (11) );        // 会員番号を表示
```

▪ 以下に示すのは、*nango* が参照するインスタンスの名前・会員番号・年齢を、*print* によって表示するプログラムである。

```
nango  (12) ;
```

▪ 以下に示すプログラムについて、コンパイルエラーとなる行には×を、エラーとならない行には○を記入せよ。

なお、 (19) ～ (24) には、*name*，*no*，*age* の宣言から **private** を取り除いた場合の挙動を同様に○×で記入すること。

```
 (13)  nango_age = 50;          (19)
 (14)  int a = nango_age;       (20)
 (15)  nango.age = 50;          (21)
 (16)  int b = nango.age;       (22)
 (17)  age.nango = 50;          (23)
 (18)  int c = age.nango;       (24)
```

▪ クラス *Member* の宣言から (2) の箇所を丸ごと削除すると、 (25) 。

▶ (25) の選択肢：(a)コンパイルできなくなる
(b)インスタンスを生成できなくなる
(c)インスタンス生成時に引数を渡せなくなる

▪ クラス *Member* の宣言から (2) の中の 3 箇所の **this.** を削除すると、 (26) 。

▶ (26) の選択肢：(a)コンパイルできなくなる
(b) *name*，*no*，*age* には不定値が入れられる
(c) *name* は空参照となり、*no*，*age* は 0 となる

▪ 以下のプログラムは、クラス *Abc* と、それを利用するクラス *AbcTester* で構成されている。二つのクラスは、AbcTester.java という名前の単一ソースファイルに格納されている。

```
(27) Abc {
   private int a;
   private int b;
   private int c;

   // コンストラクタ
   (28)(int a, int b, int c) {
      (29).a = a;
      (29).b = b;
      (29).c = c;
   }

   // a, b, cの和を取得
   (30) getSum() {
      return a + b + c;
   }

   // a, b, cの値を表示して改行
   (31) print() {
      System.out.printf("(%d, %d, %d)\n", a, b, c);
   }
}

(32) AbcTester {
   public (33) void (34)(String[] args) {
      Abc x = (35)(1, 2, 3);
      Abc y = (36)(2, 4, 6);

      int xs = (37);      // xのa, b, cの和
      int ys = (38);      // yのa, b, cの和

      System.out.print("x  : ");
      (39);                          // xのメソッドprintを呼び出す
      System.out.println("xs : " + xs);

      System.out.print("\ny  : ");
      (40);                          // yのメソッドprintを呼び出す
      System.out.println("ys : " + ys);

   }
}
```

```
x  : (1, 2, 3)
xs : 6

y  : (2, 4, 6)
ys : 12
```

▪ ソースファイル AbcTester.java のコンパイルは、(41) によって行う。

▶ (41) の選択肢：(a) javac Abc.java と javac AbcTester.java の両方の実行
(b) javac Abc.java の実行
(c) javac AbcTester.java の実行

▪ クラス *AbcTester* 内の **main** メソッドの実行は、(42) によって行う。

▶ (42) の選択肢：(a) java AbcTester + Abc の実行
(b) java Abc の実行
(c) java AbcTester の実行

▪ 左ページのクラス *Abc* が Abc.java に格納され、クラス *AbcTester* が AbcTester.java に格納されているとする（すなわち、個々のクラスが個別のソースファイルに格納されているとする）。両方のソースファイルがコンパイルずみであるときに、クラス *AbcTester* の内の **main** メソッドの実行は、［ (43) ］によって行う。

▶ ［ (43) ］の選択肢：(a) java AbcTester + Abc の実行
(b) java Abc の実行　(c) java AbcTester

▪ *x* や *y* に所属する a や *b* や *c* のことを、“［ (44) ］変数”と呼ぶ。

▪ フィールド a，b，c に与えられているアクセス性は“［ (45) ］アクセス”である。宣言に **private** が指定されていなかったら、そのアクセス性は“［ (46) ］アクセス”となる。

▪ フィールドを非公開として外部から保護することを“データ［ (47) ］”という。これを行った上で、メソッドとフィールドをうまく連係させることを［ (48) ］という。

▪ インスタンスの状態を表すフィールドは、カタカナ 4 文字で［ (49) ］とも呼ばれる。

▪ クラス内で **static** を付けずに宣言されたメソッドは、個々のインスタンスに所属するため、“［ (50) ］メソッド”と呼ばれる。インスタンスに対してメソッドを呼び出すことは、そのインスタンス（オブジェクト）に［ (51) ］を送る、と表現される。

▪ インスタンス内の個々のフィールドやメソッドをアクセスするための演算子 . の名称は［ (52) ］演算子である。

▪ コンストラクタの返却型は［ (53) ］。生成ずみのインスタンスに対してコンストラクタを呼び出すことは［ (54) ］。インスタンスに対して、コンストラクタが 1 個も定義されていないクラスには、“［ (55) ］コンストラクタ”がコンパイラによって自動的に定義される。［ (55) ］コンストラクタは、［ (56) ］。

▶ ［ (53) ］の選択肢：(a) **void** である　(b)宣言できない
(c)自分自身と同じクラス型である

▶ ［ (54) ］の選択肢：(a)できる　(b)できない

▶ ［ (56) ］の選択肢：(a)自分自身と同じクラス型の仮引数を 1 個だけ受け取る
(b)仮引数を受け取らない

▪ クラスのフィールドと同名の仮引数や局所変数をもつコンストラクタおよびインスタンスメソッドは、［ (57) ］。

▶ ［ (57) ］の選択肢：(a)コンパイルできなくなる
(b)仮引数・局所変数の名前が隠される
(c)フィールドの名前が隠される