錬成問題

▪ クラスの［ (1) ］を行うことによって、既存クラスの資産を継承したクラスを作ることができる。［ (1) ］元のクラスを“［ (2) ］クラス”と呼び、［ (1) ］によって作られたクラスを“［ (3) ］クラス”と呼ぶ。なお、複数クラスの資産を継承するために、複数のクラスからの［ (1) ］を行うことは［ (4) ］。

明示的な［ (1) ］を行わないクラスは、［ (5) ］パッケージに所属する［ (6) ］クラスから［ (1) ］したクラスとなる。

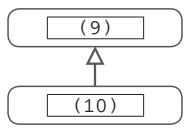
▶ ［ (4) ］の選択肢：(a)できる　(b)できない

▪ コンストラクタを一つも定義していないクラスには、コンパイラによって“［ (7) ］コンストラクタ”が自動的に定義される。クラス *X* に対する［ (7) ］コンストラクタは以下のようになる。

```
X() { [ (8) ] }
```

▪ 右ページに示すのは、会員クラス *Member*、優待会員クラス *SpecialMember*、それらをテストするクラス *MemberTester* である。

以下に示すのは、これら三つのクラスのうちの二つのクラスのクラス階層図である。



▪ クラス *SpecialMember* にとって、親クラス *Member* は［ (11) ］クラスであり、クラス *Member* にとって、子クラス *SpecialMember* は［ (12) ］クラスである。なお、クラス *Member* は、［ (13) ］クラスの子クラスである。

クラス *Member* とクラス *SpecialMember* との間に成立しているのが、“［ (14) ］の関係”であり、『クラス *SpecialMember* は［ (15) ］のクラス *Member* である。』と表現する。ここで逆の関係『クラス *Member* は［ (15) ］のクラス *SpecialMember* である。』は［ (16) ］。

▶ ［ (16) ］の選択肢：(a)成立する　(b)成立しない

▪ **1** のメソッド *print* の呼出しでは、*Member* 型のクラス型変数が参照するインスタンスの型に応じて、クラス *SpecialMember* のメソッド *print* と、クラス *Member* のメソッド *print* のいずれかが呼び出されることになる。

いずれのメソッドを呼び出すかの決定は、［ (17) ］時に行われる。このような、呼び出すべきメソッド決定のメカニズムを、［ (18) ］あるいは［ (19) ］と呼ぶ。

▶ ［ (17) ］の選択肢：(a)コンパイル　(b)実行

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

   // コンストラクタ
   public Member(String name, int no, int age) {
      this.name = name;  this.no = no;  this.age = age;
   }

   // 名前を取得する（nameのゲッタ）
   public String getName() {
      return name;
   }

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

```
//--- 優待会員クラス ---//
public class SpecialMember [ (20) ] Member {
   private String privilege;                // 特典

   // コンストラクタ
   public SpecialMember(String name, int no, int age, [ (21) ] privilege) {
      [ (22) ](name, no, age);
      [ (23) ].privilege = privilege;     // 特典
   }

   // 表示（会員番号・名前・年齢・特典）
   @[ (24) ] [ (25) ] void print() {
      [ (26) ].print();
      System.out.println("特典：" + privilege);
   }
}
```

```
//--- 会員クラスのテスト ---//
public class MemberTester {

   public static void main(String[] args) {
      Member[] m = {
         [ (27) ] Member("橋口", 101, 27),
         [ (27) ] SpecialMember("黒木", 102, 31, "会費無料"),
         [ (27) ] SpecialMember("松野", 103, 52, "会費半額免除"),
      };

      // 配列mの全要素に対してメソッドprintを呼び出す
      for ( [ (28) ] k : m) {
         k.print();  ←1
         System.out.println();
      }
   }
}
```

```
No.101：橋口（27歳）
No.102：黒木（31歳）
特典：会費無料
No.103：松野（52歳）
特典：会費半額免除
```

- クラス *SpecialMember* 中の [(25)] は省略 [(29)]。
  - ▶ [(29)] の選択肢：(a)できる　(b)できない

▪ サブクラスで継承されるものに○を、継承されないものに×を記入せよ（いずれも明示的に **private** 宣言されていないとする）。

○ フィールド … (30)　　○ コンストラクタ … (31)
○ メソッド … (32)　　○ インスタンス初期化子 … (33)
○ 静的初期化子 … (34)

▪ 上位クラスのメソッドと同じ形式のメソッドに対して、下位クラスにおいて独自の定義を与えて上書きすることを“(35)する”と呼ぶ。このようなメソッドの宣言には、**@Override** という(36)を与えるとよい。(36)は、人間とコンパイラに伝える注釈である。

なお、クラスやメソッドの仕様変更などによって、利用することが推奨されなくなったクラスやメソッドには、(37)という(36)を与えて宣言する。

メソッドを(35)する際は、そのメソッドに対して、上位クラスのメソッドと同等もしくは(38)アクセス制限をもつ修飾子を与えなければならない。

▶ (38)の選択肢：(a)弱い　(b)強い

▪ 上位クラスにおいて **public** 付きで宣言されたメソッドを(35)する際に与える修飾子は(39)であり、**private** 付きで宣言されたメソッドを(35)する際に与える修飾子は(40)であり、**protected** 付きで宣言されたメソッドを(35)する際に与える修飾子は(41)であり、アクセス修飾子なしで宣言されたメソッドを(35)する際に与える修飾子は(42)である。

※(39)～(42)は、該当する修飾子が複数あれば、すべてを記入すること。

▪ クラスのメンバとは、五十音順に(43)、(44)、(45)、(46)の総称である。

▪ スーパークラスに所属する非公開でないメンバをアクセスする式は、“(47).メンバ名”である。

▪ そのクラスからの派生クラスを作るべきでないクラスには、キーワード(48)を付けて宣言するとよい。そのクラスから派生したクラスで(35)されるべきでないメソッドは、キーワード(49)を付けて宣言するとよい。

▪ 既存のプログラムに対して必要最低限の追加・修正だけで新しいプログラムを作成する手法を“(50)プログラミング”と呼ぶ。

▪ クラス型変数が、派生関係にある様々なクラス型のインスタンスを参照できることを(51)あるいは(52)と呼ぶ。

▪ オブジェクト指向の三大要素は、五十音順に(53)、(54)、(55)である。

▪ 以下に示すのは、クラス X と、X から派生したクラス Y のコンストラクタである。

```
X(int x) {  a  this.x = x;  b }
```

```
Y(int x, int y) {  c  super(x);  d  this.y = y;  e  }
Y(int x, int y, int z) {  this.x = x; this.y = y; this.z = z;  }
```

▪ クラス X にインスタンス初期化子が定義されている場合、それが呼び出されるタイミングは a、b のうち (56) である。クラス Y にインスタンス初期化子が定義されている場合、それが呼び出されるタイミングは c、d、e のうち (57) である。

▪ 正しい記述には○を、誤った記述には×を記入せよ。

(58) …クラス X に静的初期化子が定義されていなければコンパイルエラーとなる。

(59) …ここに示すコンストラクタ以外に、引数を受けとらないコンストラクタがクラス X に多重定義されていなければ、クラス X はコンパイルエラーとなる。

(60) …ここに示すコンストラクタ以外に、引数を受けとらないコンストラクタがクラス X に多重定義されていなければ、クラス Y はコンパイルエラーとなる。

(61) …クラス X のスーパークラスが引数を受け取るコンストラクタをもっていれば、クラス X はコンパイルエラーとなる。

(62) …クラス Y から派生したサブクラス Z に、明示的に宣言されたコンストラクタがなければ、クラス Z はコンパイルエラーとなる。

▪ 正しい宣言や文には○を、誤った宣言や文には×を記入せよ。

```
(63)  X a = new X(1);
(64)  Y b = new X(1);
(65)  X c = new Y(1, 2);
(66)  Y d = new Y(1, 2);
(67)  X e = (Y)new X(1);
(68)  Y f = (X)new Y(1, 2);
```

▪ 変数 v が X 型のクラス型変数であるとする。v の参照先がクラス X 型のインスタンスであるとき、v **instanceof** X が生成する値は (69) であり、v **insanceof** Y が生成する値は (70) である。v の参照先がクラス Y 型のインスタンスであるとき、v **instanceof** X が生成する値は (71) であり、v **insanceof** Y が生成する値は (72) である。

▪ スーパークラス型からサブクラス型への型変換のことを“ (73) 変換”あるいは“ (74) キャスト”と呼び、サブクラス型からスーパークラス型への型変換のことを“ (75) 変換”あるいは“ (76) キャスト”と呼ぶ。