

錬成問題

▪ フィールドの値を取得するメソッドを と呼び、設定するメソッドを と呼ぶ。両者の総称が である。

▪ 代入もしくは初期化によって、クラス型変数の値をコピーすると、。また、メソッドの引数としてクラス型変数をやりとりする際は、。

▶ 共通の選択肢：(a)全フィールドの値がコピーされる (b)参照先がコピーされる

▪ クラス型変数の値を等価演算子 == または != によって比較すると、。

▶ 共通の選択肢：(a)全フィールドの値の等価性が判断される
(b)クラス型変数の参照先の等価性が判断される

▪ コンストラクタは多重定義 。メソッドは多重定義 。

▶ 共通の選択肢：(a)できる (b)できない

▪ 同一クラス型の引数を受け取って、その全フィールドの値をコピーするコンストラクタは、“ コンストラクタ”と呼ばれる。

▪ クラスには、インスタンスの現在の状態を簡潔な文字列表現で返却する、以下の形式のメソッドを定義するとよい。

() { /*… メソッド本体 …*/ }

というのも、このメソッドは、『クラス型変数+文字列』および『文字列+クラス型変数』の演算において、自動的に呼び出されるからである。

▪ `this(...)` は、 であり、 行える。

▶ の選択肢：(a)同一クラスに所属する他のコンストラクタの呼出し
(b)コンストラクタである自分自身の呼出し
(c)Java が提供する標準的な API の呼出し

▶ の選択肢：(a)メソッドの先頭でのみ (b)コンストラクタの先頭でのみ
(c)任意の場所で (d)メソッドの外部で

▪ クラス型の配列を生成すると、全要素が で初期化される。

▪ 不用意に 型フィールドの値を返却してはならない。返却された値を通じて、外部から間接的に値が書きかえられてしまうからである。

▪ 右ページに示すのは、単一の `int` 型フィールドをもつクラス `Int` と、それを利用するプログラムである。

```

(17) java.(18).Scanner;
//--- 整数クラス ---//
public class Int {
    private int v;
    public Int(int v)    { this.v = v; }        // コンストラクタ
    public int getV()    { return (19); }      // (1)
    public void setV(int v) { (20) = v; }      // (2)
}
//--- 整数クラスのテスト ---//
public class IntTester {
    (21) int compare((22) x, (23) y) {
        if ((24) > (25))                // xのほうが大きい
            return 1;
        else if ((26) < (27))           // xのほうが小さい
            return -1;
        return 0;
    }

    public static void main(String[] args) {
        Scanner stdin = new Scanner((28));

        System.out.print("a = "); int ta = stdin.nextInt();
        System.out.print("b = "); int tb = stdin.nextInt();
        Int a = (29)(ta);
        Int b = (30)(tb);

        int balance = compare((31), (32));
        if (balance == 1)
            System.out.println("aのほうが大きい。");
        else if (balance == -1)
            System.out.println("bのほうが大きい。");
        else
            System.out.println("aとbは同じ。");
    }
}

```

```

a = 10
b = 20
bのほうが大きい。

```

■ このソースプログラムから、ある語句を削除しない限り、コンパイルエラーとなる。その語句と、コンパイルエラーとなる理由を示せ。… (33)

■ 以下に示すのは、要素数が5のInt型配列aの宣言である。なお、各要素のフィールドvの値が先頭から順に1, 2, 3, 4, 5となるように初期化するものとする。

```
(34) a = (35);
```

■ 以下に示すのは、要素数が5のInt型の2次元配列bである。行数は2であり、列数が3である0行目の各要素のフィールドvの値が先頭から順に1, 2, 3、列数が4である1行目の各要素のフィールドvの値が先頭から順に5, 6, 7, 8となるように代入するものとする。

```
(36) b = (37);
b[0] = new (38);
b[1] = new (39);
```

- 以下に示すのは、2次元座標クラス、円クラス、それらのクラスをテストするプログラムである。

```
//--- 2次元座標クラス ---//
public (40) Point2D {
    private int x = 0;          // X座標
    private int y = 0;          // Y座標

    public Point2D() { }
    public Point2D(int x, int y) { set(x, y); }
    public Point2D(Point2D p) { this((41), (42)); }

    public int getX() { return x; }
    public int getY() { return y; }

    public void setX(int x) { (43) = x; }
    public void setY(int y) { (44) = y; }
    public void set(int x, int y) { (45)(x); (46)(y); }

    public String toString() { return "(" + x + ", " + y + ")"; }
}
```

```
//--- 円クラス ---//
public (47) Circle {
    private Point2D center;     // 中心の座標
    private int radius = 0;     // 半径

    public Circle() { center = new (48); }
    public Circle(Point2D c, int radius) {
        center = (49); this.radius = radius;
    }

    public Point2D getCenter() { return new (50)((51)); }
    public int getRadius() { return radius; }

    public void setCenter(Point2D c) {
        center.set((52), (53));
    }
    public void setRadius(int radius) { (54) = radius; }

    public String toString() {
        return "中心座標：" + center.(55) + " 半径：" + radius;
    }
}
```

```
//--- 円と座標のテスト ---//
public class CircleTester {
    public static void main(String[] args) {
        Point2D[] p = (56) {
            (57) Point2D(3, 7), (58) Point2D(4, 6)
        };
        Circle c1 = new Circle();
        Circle c2 = new Circle(new Point2D(10, 15), 5);

        for (int i = 0; i < p.length; i++)
            System.out.println("p[" + i + "] = " + (59));

        c1.setRadius(10); // 半径を10に変更
        System.out.println("c1 = " + (60));
        System.out.println("c2 = " + (61));
    }
}
```

```
p[0] = (3,7)
p[1] = (4,6)
c1 = 中心座標：(0,0) 半径：10
c2 = 中心座標：(10,15) 半径：5
```

▪ 以下に示すのは、自分自身の座標が、引数で与えられた座標 p と等しい (X座標の値と Y座標の値がともに等しい) かどうかを判定するインスタンスメソッドである。

なお、仮引数 p に受け取るのは、 である。

- ▶ の選択肢：(a)座標クラス *Point2D* 型のインスタンスそのもの
(b)座標クラス *Point2D* 型のインスタンスへの参照

```
boolean equalTo( p) {
    return ;
}
```

▪ 以下に示すのは、自分自身の円が、引数で与えられた円 c と等しい (中心の座標と半径の値がともに等しい) かどうかを判定するインスタンスメソッドである。

```
boolean equalTo( p) {
    return ;
}
```

▪ 以下に示すのは、二つの円 $c1$ と $c2$ が等しい (中心の座標と半径の値がともに等しい) かどうかを判定する、クラス *Circle* の外部で定義するメソッドである。

```
static boolean equal( c1,  c2) {
    return ;
}
```

▪ 以下に示すのは、要素数 2 の円クラス *Circle* 型 a の配列の宣言である。なお、最初の要素の座標が (0, 0) で半径が 5 となるように初期化して、2 番目の要素の座標が (10, 10) で半径が 8 となるように初期化する。

```
 a = ;
```

▪ 以下に示すのは、*Circle* 型の変数 a が参照する円の中心座標の X 座標と Y 座標を表示するプログラムである。

```
System.out.println("aの中心のX座標 = " + );
System.out.println("aの中心のY座標 = " + );
```

▪ 使い捨てのものやテスト的なものでない限り、クラスやメソッドには を付けて宣言するとよい。パッケージを越えて利用できるからである。

▪ クラスのフィールドが他のクラス型となっているとき、“ の関係” が成立する。