オブジェクト指向プログラミング H28 年度 後期中間試験(2016.11.30 重村 哲至)

IE5 ＿＿＿番 氏名 模範解答

別紙のプログラムについて答えなさい。

1. プログラム中の空欄に適切なプログラムの断片を書きなさい。(4 点× 10 問=40 点)

(1) /*** (A) ***/ 部分に適切なプログラム

```
view.setModel(model);
```

(2) /*** (B) ***/ 部分に適切なプログラム

```
model.put(1,0);
```

(3) /*** (C) ***/ 部分に適切なプログラム

```
extends JPanel
```

(4) /*** (D) ***/ 部分に適切なプログラム

```
model.get(x,y)
```

(5) /*** (E) ***/ 部分に適切なプログラム

```
model.getKekka()
```

(6) /*** (F) ***/ 部分に適切なプログラム

```
protected void
```

(7) /*** (G) ***/ 部分に適切なプログラム

```
private int
```

(8) /*** (H) ***/ 部分に適切なプログラム

```
private void
```

(9) /*** (I) ***/ 部分に適切なプログラム

```
public Masume
```

(10) /*** (J) ***/ 部分に適切なプログラム

```
public Kekka
```

2. リスト１の３３行～３８行は何をする処理か説明しなさい。(20 点)

   まず、何クラスのどのようなインスタンスを作っているのか？

   次に、それをどのようにしているのか？

ActionListener の actionPerformed() をオーバーライド (5 点) した（内部）無名クラスのインスタンスを作る (5 点)。そのインスタンスを btnNw にアクションリスナとして登録する (5 点)。なお、オーバーライドのした actionPerformed() は、model と view にメッセージを送るものである (5 点)。

3. 以下を読んで問に答えなさい。(20 点)

   リスト４はリスト１のプログラム改良したものです。３並べアプリ専用の JButton クラスを作って使用したので、リスト４ではリスト１の３３行～３８行に相当する処理がとても簡単になりました。SanMoku クラスのインスタンス変数 model と view にアクセスするために、SanMoku クラスの内部クラスとして SanButton クラスを記述しています。(Java では外側クラスの変数に、リスト４の１３、１４行のような書き方でアクセスできます。)

   一方で、リスト１の３３行～３８行の処理を簡単にするために、普通のクラスを作る方法も考えられます。initialize() 実行時には、まだ Model と View のインスタンスがありませんので、SanButton のコンストラクタに Model と View を渡すことができません。そこで、代わりに SanMoku クラスに次のようなメソッド delegate() を追加し、SanMoku クラスのインスタンスを SanButton のコンストラクタに渡すことにしました。

```
1 public void delegate(int x, int y) {
2     model.put(x, y);
3     view.repaint();
4 }
```

   次のクラス図を参考に SanButton クラスを Java 言語で記述しなさい。但し、import は省略してよい。

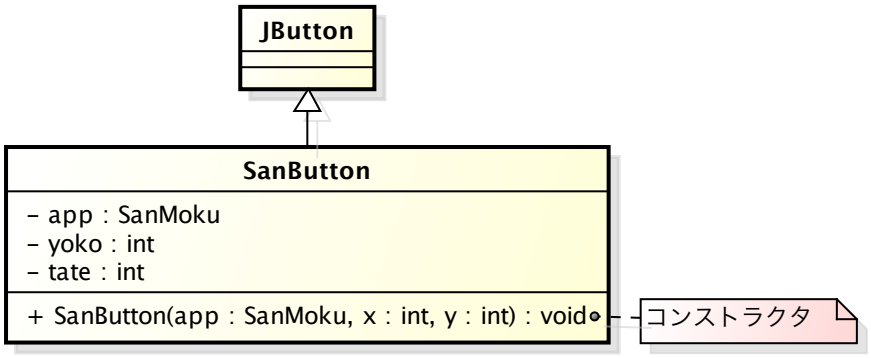

図 1 SanButton クラス

```
import javax.swing.JButton;

public class SanButton extends JButton {
  private SanMoku app;
  private int yoko;
  private int tate;

  SanButton(SanMoku app, int x, int y) {
    super("◯");
    this.app = app;
    this.yoko = x;
    this.tate = y;
    addActionListener(new ActionListener() {
      public void actionPerformed(
                              ActionEvent e) {
        app.delegate(yoko, tate);
      }
    });
    setPreferredSize(new Dimension(29, 29));
  }
}
```

4. 前問の SanButton クラスを使用する時、リスト４の３５行はどのように書くべきか、以下に書きなさい。(5 点)

```
SanButton btnNw = new SanButton(this,0,0);
```

5. 別紙の図３を見て答えなさい。(3 点× 5 問=15 点)

   (a) SanMoku クラスと SanMokuModel クラスの関連を何と言うか。

   合成集約 (composition)

   (b) JPanel クラスと SanMokuView クラスの関連を何と言うか。

   継承、汎化 (generalization)

   (c) JPanel クラスと SanMokuView クラスで機能の多いのはどちらか。簡単に理由も書きなさい。

   SanMokuView クラス
   サブクラスはスーパークラスの全ての機能を持った上で何か変更や追加を行ったものであるので、一般にサブクラスの方が機能が多い。

   (d) アプリの画面にゲームを最初からやり直す RESET ボタンを追加します。現在の SanMokuModel クラスのままではうまく行きません。SanMokuModel クラスをどのように変更すべきか答えなさい。

   モデルの状態を最初に戻すメソッドが必要である。そのようなメソッドを新しく追加するか、reset() メソッドを public にする必要がある。

   (e) SanMokuModel クラスを変更することなく、継承を用いて RESET ボタンに対応させることは可能か？ 理由も含めて答えなさい。(ヒント：private なメソッドはオーバーライドできない。private なメンバーにはサブクラスからアクセスできない。)

   不可能である。
   プロパティも reset() メソッドも private なので、サブクラスがプロパティや reset() にアクセスすることも、reset() をオーバーライドすることもできない。よって、サブクラスに機能を実装できない。

図２に実行例を示す３目並べアプリは「人間」と「コンピュータ」で３目並べを戦うアプリケーションプログラムです。必ず人間が先手（「○」）です。

人間は画面の右にある９個（３×３個）のボタンで「○」を書く場所を指定します。コンピュータは乱数を使った思考ルーチンで「×」を書く場所を決めます。

図３にクラス図を示します。次ページからのソースコード中で、※１から※２、※３から※４は、実装の都合で追加した変数やメソッドなのでクラス図には書いてありません。また、`SanMoku` クラスのクラス図は、Eclipse が自動的に作る部分などをかなり省略しています。関連端名（-model や -view）は、プロパティを表してることにしますので注意して下さい。

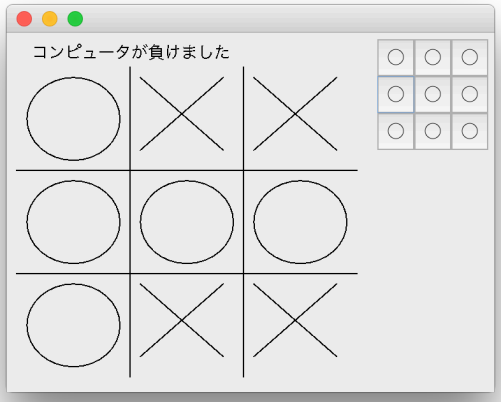

図２　３目並べアプリの実行例

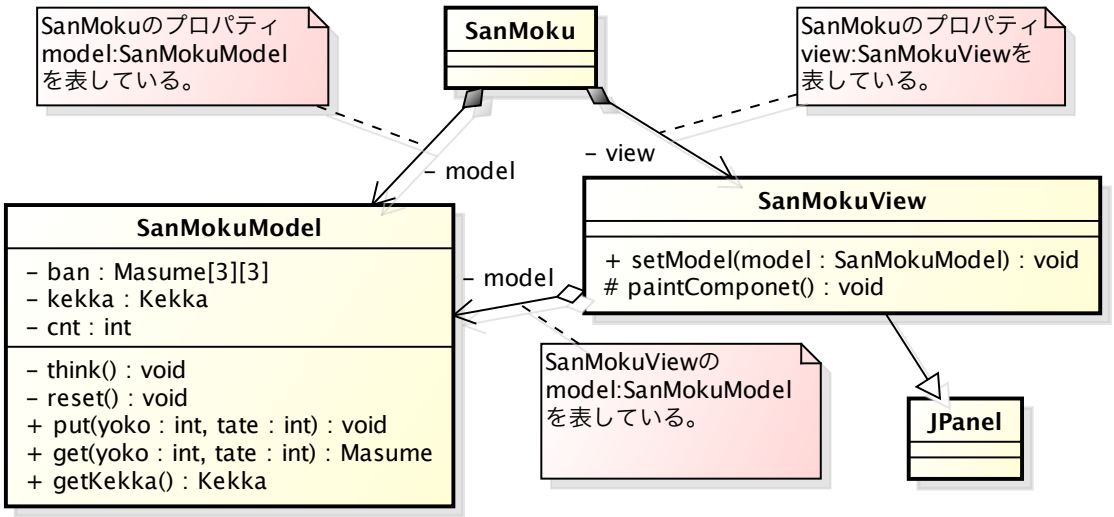

図３　３目並べアプリのクラス図

リスト 1: SanMoku.java

```
1  ... import 省略 ...
2  public class SanMoku {
3      private JFrame frame;          // Eclipse が frame をここに書いた
4      private SanMokuModel model;    // ３目並べの盤面と思考ルーチン(モデル)
5      private SanMokuView view;      // ３目並べの表示(ビュー)
6
7      public static void main(String[] args) {
8          ... Eclipse が作るいつもの main の内容なので省略 ...
9      }
10
11     public SanMoku() {
12         initialize();
13         model = new SanMokuModel();   // モデルオブジェクトを作って
14         /*** (A) ***/                 // ビューにモデルを登録する
15     }
16
17     private void initialize() {
18         frame = new JFrame();
19         frame.setBounds(100, 100, 383, 304);
20         frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
21         JPanel pnlCtrl = new JPanel();
22         frame.getContentPane().add(pnlCtrl, BorderLayout.EAST);
23         JPanel pnlBtn = new JPanel();
24         pnlCtrl.add(pnlBtn);
25         pnlBtn.setLayout(new GridLayout(0, 3, 0, 0));
26         /* ボタンの配置とアクションリスナの登録
27          *       0     1     2
28          *   0 btnNw btnN  btnNe    <-- ９個のボタンの名前と配置
29          *   1 btnW  btnC  btnE
30          *   2 btnSw btnS  btnSe
31          */
32         JButton btnNw = new JButton("◯");                  // ボタン Nw
33         btnNw.addActionListener(new ActionListener() {
34             public void actionPerformed(ActionEvent e) {
35                 model.put(0, 0);                           // (0,0) に'◯'を置く
36                 view.repaint();                            // ビューに再描画依頼
37             }
38         });
39         btnNw.setPreferredSize(new Dimension(29, 29));
40         pnlBtn.add(btnNw);
41         JButton btnN = new JButton("◯");                   // ボタン N
42         btnN.addActionListener(new ActionListener() {
43             public void actionPerformed(ActionEvent e) {
44                 /*** (B) ***/                             // (1,0) に'◯'を置く
45                 view.repaint();                            // ビューに再描画依頼
46             }
47         });
48         btnN.setPreferredSize(new Dimension(29, 29));
49         pnlBtn.add(btnN);
50
51         ... btnNe, btnW, btnC, btnE, btnSw, btnS, btnSe が続く ...
52
53         view = new SanMokuView();                          // ビューを作って画面に配置
54         frame.getContentPane().add(view, BorderLayout.CENTER);
55     }
56 }
```

リスト 2: SanMokuView.java

```
... import 等省略 ...
public class SanMokuView /*** (C) ***/ {
    private SanMokuModel model;                  // モデル

    public void setModel(SanMokuModel model) {
        this.model = model;
    }

    // 描画に使用する一時的な変数 ---------- ※ l
    private int ww;                              // 縦線の間隔
    private int hh;                              // 横線の間隔
    private int xs[] = new int[4];               // 縦横線の座標
    private int ys[] = new int[4];               // 縦横線の座標
    private int dw;                              // 幅方法の余白
    private int dh;                              // 高さ方法の余白

    private void calc() {                        // 必要な座標を計算する
        int w = getWidth();                      // パネルの幅
        int h = getHeight();                     // パネルの高さ
        ww = (int)(w * 0.94 / 3.0);              // l マスの幅
        hh = (int)(h * 0.87 / 3.0);              // l マスの高さ
        dw = (int)(w * 0.03);                    // 横方向の余白
        dh = (int)(h * 0.03);                    // 高さ方向の余白
        int sh=(int)(h*0.07);                    // 文字列表示エリア
        for (int i=0; i<4; i++) {                // マス目の交点座標
            xs[i]= getX()+dw+ww*i;
            ys[i]= getY()+dh+hh*i+sh;
        }
    }

    private void drawLines(Graphics g) {         // 縦横の線を描く
        for (int i=1; i<3; i++) {
            g.drawLine(xs[0], ys[i], xs[3], ys[i]);
            g.drawLine(xs[i], ys[0], xs[i], ys[3]);
        }
    }

    private void drawMaruBatu(Graphics g) {      // ○と×を描く
        for (int y=0; y<3; y++) {                // 全てのマス目について
            for (int x=0; x<3; x++)
                switch ( /*** (D) ***/ ) {       // 状態を調べる
                case MARU:
                    g.drawOval(xs[x]+dw, ys[y]+dh, ww-dw*2, hh-dh*2);        // ○を描く
                    break;
                case BATU:
                    g.drawLine(xs[x]+dw,ys[y]+dh,xs[x+1]-dw*2,ys[y+1]-dh*2); // ×を描く
                    g.drawLine(xs[x]+dw,ys[y+1]-dh*2,xs[x+1]-dw*2,ys[y]+dh);
                    break;
                default:
                    break;
                }
        }
    }

    private void drawKatiMake(Graphics g) {      // 勝敗を画面に書く
        switch ( /*** (E) ***/ ) {               // 勝敗を調べる
        case KATI:
```

```
58              g.drawString("コンピュータが勝ちました", 20, 20);
59              break;
60          case MAKE:
61              g.drawString("コンピュータが負けました", 20, 20);
62              break;
63          case HIKIWAKE:
64              g.drawString("引き分けです", 20, 20);
65              break;
66          default:
67              break;
68          }
69      }
70      // ------------------------------------- ※2
71
72      @Override
73      /*** (F) ***/ paintComponent(Graphics g) {
74          super.paintComponent(g);
75          if (model==null) return;
76          calc();
77          drawLines(g);
78          drawMaruBatu(g);
79          drawKatiMake(g);
80      }
81  }
```

リスト 3: SanMokuModel.java

```
1   import java.util.Random;
2   enum Masume {NASI, MARU, BATU};             // マス目の状態型:無し、◯、×
3   enum Kekka  {FUMEI, KATI, MAKE, HIKIWAKE};  // 勝敗型:不明、勝ち、負け、引き分け
4
5   public class SanMokuModel {                 // ３目並べを表現するクラス(モデル)
6       private Masume[][] ban;                 // 3 × 3 のマス目
7       private Kekka kekka;                    // 勝敗
8       private Random random = new Random();   // 乱数発生器
9       /*** (G) ***/ cnt;                      // 手数
10
11      public SanMokuModel() {                 // コンストラクタ
12          ban = new Masume[3][3];             // マス目を作って
13          reset();                            // クリアする
14      }
15
16      private void reset() {                  // マス目のクリア
17          for (int y=0; y<3; y++)
18              for (int x=0; x<3; x++)
19                  ban[x][y] = Masume.NASI;    // 空白
20          kekka = Kekka.FUMEI;                // 勝敗不明
21          cnt = 0;                            // 手数はゼロ
22      }
23      // --------------------------------- ※3
24      private Kekka katiMake(Masume m) {      // どっちが勝ったか判定する
25          return (m==Masume.MARU)?Kekka.MAKE:Kekka.KATI;
26      }
27
28      private void hantei() {                 // 盤面全体を調べて勝敗を判定する
29          for (int i=0; i<3; i++) {           // 縦横のチェック
30              if (ban[i][0]!=Masume.NASI &&
31                  ban[i][0]==ban[i][1] &&
```

```
32                  ban[i][1]==ban[i][2]) {
33                  kekka=katiMake(ban[i][0]);
34                  return;
35              }
36              if (ban[0][i]!=Masume.NASI &&
37                  ban[0][i]==ban[1][i] &&
38                  ban[1][i]==ban[2][i]) {
39                  kekka=katiMake(ban[0][i]);
40                  return;
41              }
42          }
43          if (ban[1][1]!=Masume.NASI &&       // 左上から右下の斜めチェック
44              ban[0][0]==ban[1][1] &&
45              ban[1][1]==ban[2][2]) {
46              kekka=katiMake(ban[1][1]);
47              return;
48          }
49          if (ban[1][1]!=Masume.NASI &&       // 左下から右上の斜めチェック
50              ban[2][0]==ban[1][1] &&
51              ban[1][1]==ban[0][2]) {
52              kekka=katiMake(ban[1][1]);
53              return;
54          }
55          if (cnt>=3*3) kekka=Kekka.HIKIWAKE; // 手数が９になっていたら引き分け
56      }
57      // ---------------------------------- ※4
58      /*** (H) ***/ think() {                 // 思考ルーチン(乱数版)
59          cnt++;
60          hantei();
61          if (kekka!=Kekka.FUMEI) return;
62          for (;;) {                          // 打つ場所を乱数で決める
63              int rx = random.nextInt(3);
64              int ry = random.nextInt(3);
65              if (ban[rx][ry]==Masume.NASI) {
66                  ban[rx][ry]=Masume.BATU;
67                  break;
68              }
69          }
70          cnt++;
71          hantei();
72      }
73
74      public void put(int x, int y) {         // 人間が (x,y) に打つ
75          if (kekka!=Kekka.FUMEI ||
76              ban[x][y]!=Masume.NASI) return;
77          ban[x][y] = Masume.MARU;
78          think();
79      }
80
81      /*** (I) ***/ get(int x, int y) {       // (x,y) のマス目の状態を返す
82          return ban[x][y];
83      }
84
85      /*** (J) ***/ getKekka() {              // 勝敗を返す
86          return kekka;
87      }
88  }
```

リスト 4: SanMoku.java（SanButton を内部クラスにした）

```
1  ... import 省略 ...
2  public class SanMoku {
3    private JFrame frame;                // Eclipse が frame をここに書いた
4    private SanMokuModel model;          // ３目並べの盤面と思考ルーチン(モデル)
5    private SanMokuView view;            // ３目並べの表示(ビュー)
6
7    // ３目並べ専用の JButton クラス（SanMoku クラスの内部クラスとして実装）
8    class SanButton extends JButton {
9      SanButton(final int x, final int y) {
10       super("◯");
11       addActionListener(new ActionListener() {
12         public void actionPerformed(ActionEvent e) {
13           SanMoku.this.model.put(x, y);  // 外側のクラスのインスタンス変数 model を使用できる
14           SanMoku.this.view.repaint();   // 外側のクラスのインスタンス変数 view を使用できる
15         }
16       });
17       setPreferredSize(new Dimension(29, 29));
18     }
19   }
20
21   ... 途中省略 ...
22
23   private void initialize() {
24     frame = new JFrame();
25     frame.setBounds(100, 100, 383, 304);
26     frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
27
28     JPanel pnlCtrl = new JPanel();
29     frame.getContentPane().add(pnlCtrl, BorderLayout.EAST);
30
31     JPanel pnlBtn = new JPanel();
32     pnlCtrl.add(pnlBtn);
33     pnlBtn.setLayout(new GridLayout(0, 3, 0, 0));
34
35     SanButton btnNw = new SanButton(0,0);              // ボタン Nw
36     pnlBtn.add(btnNw);                                 // こんなに簡単になる
37
38     SanButton btnN = new SanButton(1,0);               // ボタン N
39     pnlBtn.add(btnN);                                  // こんなに簡単になる
40
41     ... btnNe, btnW, btnC, btnE, btnSw, btnS, btnSe が続く ...
42
43     view = new SanMokuView();                          // ビューを作って画面に配置
44     frame.getContentPane().add(view, BorderLayout.CENTER);
45   }
46 }
```