

## 第八回 演習問題（コレクション）

## 諸注意

- 「Kadai.java」, 「MyComparator.java」の2ファイルを Web から提出する。
- 課題3以降は、できた人から教員もしくは TA を呼び、その場でチェックを受ける。
- コピペ発覚時は見せた側も見せてもらった側も両方0点とする。
- 必ず**コンパイルエラーのない状態で提出すること**（自動採点したいのでコンパイルエラーがあると、全て0点になってしまう）。
- 課題の途中で提出することになった場合、**コンパイルエラーさえ出なければ、課題の途中の状態で提出してくれて構わない**。一部のメソッドだけが実現できていない場合、コンパイルエラー出ないならばそのままの状態で提出してくれてよい。
- 主にコンソール出力で評価しているため、デバッグに用いたようなコンソール出力が残っていないように気をつけること。**基本的にコンソール出力を指定しない限りは、課題内でコンソール出力はないものとする**。
- **Package は使わないこと**（デフォルトパッケージで実装する）。Package で実装すると、自動採点がうまくいきません。

## 課題 1

1	問題設定	<p>引数として渡された List インスタンス (ArrayList でも LinkedList でもよいものとする) に対して, 色々処理するメソッドを開発してほしい. Kadai.java 内に static メソッドとして定義することとする. なお, リストの中身が正確に変更されていることは自身で確認すること.</p> <ol style="list-style-type: none"> <li>listAdder() <ul style="list-style-type: none"> <li>第一引数: List インスタンス (要素は String 型)</li> <li>第二引数: 追加するデータ数 num (int 型)</li> <li>戻り値: 処理にかかった時間[ms] (long 型)</li> <li>処理: 引数のリストに対して 0,1,2,3... と順に文字列を num 個末尾に追加していく.</li> </ul> </li> <li>listAdderO() <ul style="list-style-type: none"> <li>第一引数: List インスタンス (要素は String 型)</li> <li>第二引数: 追加するデータ数 num (int 型)</li> <li>戻り値: 処理にかかった時間[ms] (long 型)</li> <li>処理: 引数のリストに対して 0,1,2,3... と順にリストの先頭に num 個挿入していく.</li> </ul> </li> <li>listToArray() <ul style="list-style-type: none"> <li>第一引数: List インスタンス (要素は String 型)</li> <li>戻り値: String 型配列</li> <li>処理: 引数のリストの中身を一つ一つ読み出し, 新たに作成した String 型配列に代入して戻り値として配列を返す.</li> </ul> </li> </ol>
	補足	<p>System.currentTimeMillis(); で現在時刻を long 型で取得することができる. 人間が読んでも意味は分からないが, UTC 1970 年 1 月 1 日深夜零時との差を意味している.</p>
	テスト例	<p>(Main.java の main メソッドに以下をコピペする)</p> <pre> ArrayList&lt;String&gt; array = new ArrayList&lt;String&gt;(); LinkedList&lt;String&gt; link = new LinkedList&lt;String&gt;(); // 単純な追加速度を比較してみる. int num = 300000; System.out.println(Kadai.listAdder(array,num)); System.out.println(Kadai.listAdder(link,num)); // 一旦中身を削除する array.clear(); link.clear(); // 先頭に挿入する速度を比較してみる. num = 100000; System.out.println(Kadai.listAdderO(array,num)); System.out.println(Kadai.listAdderO(link,num)); // 一旦中身を削除する array.clear(); link.clear(); num = 50000; </pre>

	<pre> Kadai.listAdder(array, num); Kadai.listAdder(link, num); long start; start = System.currentTimeMillis(); Kadai.listToArray(array); System.out.println(System.currentTimeMillis()-start); start = System.currentTimeMillis(); Kadai.listToArray(link); System.out.println(System.currentTimeMillis()-start); </pre>
テスト結果 の例	(それぞれの処理にかかった時間が表示されるので、内容を考察してみるとよい)

## 課題 1 : 解答例 (Kadai.java)

```

public class Kadai {
    // ArrayListでもLinkedListでも許容しなければならないので、
    // 第一引数はList<String>型にする必要がある。
    // 今回はお情けでArrayList版とLinkedList版を
    // オーバーロード（多重定義）で実装していても良いものとした。
    public static long listAdder(List<String> list, int num){
        // 開始時間の記録
        long start = System.currentTimeMillis();
        // データをlistの末尾に追加していく
        for(int i=0;i<num;i++){
            list.add(""+i);
        }
        return System.currentTimeMillis()-start;
    }

    public static long listAdder0(List<String> list, int num){
        // 開始時間の記録
        long start = System.currentTimeMillis();
        // データをlistの先頭に挿入していく
        for(int i=0;i<num;i++){
            list.add(0, ""+i);
        }
        return System.currentTimeMillis()-start;
    }

    // Listのデータを配列に変換するメソッド
    public static String[] listToArray(List<String> list){
        // 配列の確保
        String[] res = new String[list.size()];
        // 全てのデータを配列に一つずつ格納していく
        for(int i=0;i<list.size();i++){
            res[i] = list.get(i);
        }
        return res;
    }
}

```

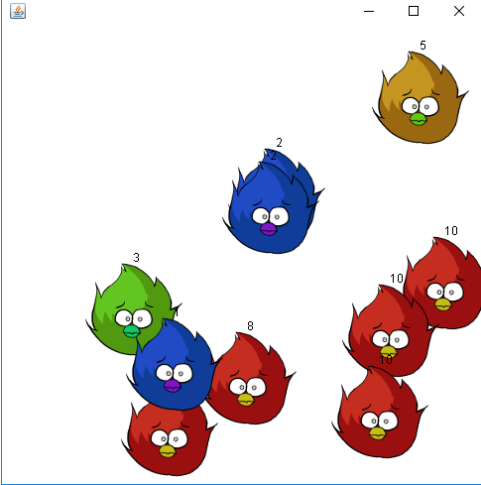
## 課題 2

2	問題設定	String 型を要素とする ArrayList を引数として渡すと長さ昇順+ABC 昇順で並べ替える sortByLength() メソッドを開発せよ。MyComparator.java 内に static メソッドで定義し、戻り値は void とする。
	ヒント	<p>Collections.sort() は Comparable を継承したインスタンスのリストをソートすることができたが、もう一つ、第二引数に Comparator を実装したインスタンスを渡すという使い方がある。</p> <p>例えば今回のように String という既に存在する型を違う方法でソートしたい場合は、MyComparator クラスを定義し、Comparator を実装するとよい。そして、sort の第二引数に new MyComparator() とする。</p> <p>ヒント 2 (白文字)：</p>
	テスト例	<pre>(Main.java の main() メソッドにて) ArrayList&lt;String&gt; list = new ArrayList&lt;String&gt;(); list.add("egg"); list.add("bat"); list.add("rose"); list.add("notebook"); list.add("chair"); list.add("fish"); list.add("bag"); MyComparator.sortByLength(list); for(String str : list){     System.out.println(str); }</pre>
	テスト結果	<pre>bag bat egg fish rose chair notebook</pre>

## 課題 2：解答例 (MyComparator.java)

```
// Stringの並び替えを行うためのMyComparatorを実装する.  
// ソート対象はString型なのでComparator<String>をimplementsする.  
public class MyComparator implements Comparator<String>{  
    // Comparatorを実装する際はcompare()をオーバーライドする.  
    @Override  
    public int compare(String o1, String o2) {  
        // 長さで大小を決定する.  
        return o1.length() - o2.length();  
    }  
    // staticメソッドなのでどこに書いても問題ないがMyComparatorクラス  
    // 内にまとめておくほうがコンパクトでよいだろう.  
    // 送られてきたlistをABC順でソートした後MyComparatorで定義した  
    // 長さ順でソートする。これにより長さ->ABC順のソートが実現される.  
    public static void sortByLength(ArrayList<String> list){  
        Collections.sort(list);  
        Collections.sort(list, new MyComparator());  
    }  
}
```

## 課題 3

	問題設定	<p>ハトヤマクリッカーは画面にランダムに表示されるハトヤマをクリックで撃退するゲームである。何もないところをクリックすると、新たなハトヤマがランダムな位置に出現する (HP10)。ハトヤマをクリックすると HP が 1 減少する (HP に応じて色が変わる：赤→黄→緑→青)。HP が 0 になるとハトヤマは消滅する。</p>
	動作画面例	 <p>図 1. 動作画面例</p>
3	ハトヤマクラス	<p>ハトヤマ 1 体を管理する Hatoyama クラスを作成する。</p> <p>フィールド</p> <ul style="list-style-type: none"> <li>hp (int 型) : 残りの HP</li> <li>x (int 型) : 表示画像の左上座標 x</li> <li>y (int 型) : 表示画像の左上座標 y</li> </ul> <p>static フィールド</p> <ul style="list-style-type: none"> <li>imgs (Image 型配列) : 各色のハトヤマ画像</li> <li>&gt; DL した zip を解凍すると hatoyama_fire0~3.png があるので、これを配列 0-3 に格納して使う。</li> </ul> <p>コンストラクタ</p> <ul style="list-style-type: none"> <li>・フィールド 3 つを初期化する。</li> <li>・imgs が null ならば画像を読み込む。</li> </ul> <p>メソッド</p> <ul style="list-style-type: none"> <li>・hp, x, y それぞれの getter メソッドを作成する。</li> <li>・boolean damage(int damage) <ul style="list-style-type: none"> <li>&gt; 第一引数の値を hp から減ずる。</li> <li>&gt; hp が 0 以下になった場合は hp を 0 とする。</li> <li>&gt; hp が 0 以下になったかどうかを boolean で返す。</li> </ul> </li> <li>・Image getImage() : 現在の hp に応じ Image を返す。 <ul style="list-style-type: none"> <li>&gt; 10-9 : 赤, 8-6 : 黄, 5-4 : 緑, 3- : 青</li> </ul> </li> </ul>

3	表示用 パネル クラス	<p>表示関連を管理する GamePanel クラスを作成する。</p> <p>フィールド</p> <p>enemies (ArrayList&lt;Hatoyama&gt;型)</p> <p>画面上に表示されるハトヤマのリスト</p> <p>コンストラクタ</p> <p>マウスリスナを登録する。</p> <p>メソッド</p> <ul style="list-style-type: none"> <li>• void generateHatoyama() <ul style="list-style-type: none"> <li>&gt;画面内のランダムな座標を初期値とするハトヤマインスタンス (HP10) を一つ作成し, enemies に追加する.</li> </ul> </li> <li>• void paintComponent(Graphics g) <ul style="list-style-type: none"> <li>&gt;enemies を全て表示する.</li> <li>&gt;各ハトヤマ表示時には画像表示に加えて右上あたりに HP も文字列で表示することとする.</li> </ul> </li> <li>• void mouseReleased(MouseEvent e) <ul style="list-style-type: none"> <li>&gt;全ハトヤマを探索し, クリック位置が重なるハトヤマに対してダメージ 1 を与える. 衝突判定後 repaint() を忘れずに.</li> <li>&gt;ハトヤマがいない場所をクリックした場合, 新たなハトヤマが誕生する.</li> </ul> </li> </ul>
	チェック ポイント	<p>途中までできて終了しそうな場合は, 次のところまででもチェックを受けて良い.</p> <ol style="list-style-type: none"> <li>1. 適当な箇所をクリックするとランダムな位置にハトヤマが登場し HP とともに表示されている.</li> <li>2. ハトヤマをクリックしたら HP が減り, 色が変わる.</li> <li>3. HP が 0 になると消滅する.</li> <li>4. (追加課題) 10 体のハトヤマが最初からいる.</li> </ol>

## 課題 3：解答例 (Hatoyama.java)

```
public class Hatoyama {
    // 指定されたとおりにフィールドを定義する。カプセル化を忘れずに。
    // imgsをstaticとしたのは、各インスタンスがimgsを保有する必要が
    // ないからである（そんなことをしてもメモリの無駄）。共有でimgsを
    // 準備し全インスタンスがアクセスすれば良いのでstaticとした。
    private static Image[] imgs = new Image[4];
    private int hp = 0;
    private int x = 0;
    private int y = 0;

    // コンストラクタは初期化処理とimgsの取得を行う。
    public Hatoyama(int hp, int x, int y){
        this.hp = hp;
        this.x = x;
        this.y = y;
        // imgs[]はnew Image[4]で4つ分の領域を確保してはいるが、
        // 個々の初期値はnullなのでチェックをしてから取得する。
        if(imgs[0] == null){
            try{
                // Imageの取得方法は何でもよい。
                // for文でも書けるが紙面の都合で直書きする。
                imgs[0]=ImageIO.read(new File("hatoyama_fire0.png"));
                imgs[1]=ImageIO.read(new File("hatoyama_fire1.png"));
                imgs[2]=ImageIO.read(new File("hatoyama_fire2.png"));
                imgs[3]=ImageIO.read(new File("hatoyama_fire3.png"));
            }catch(Exception e){
                e.printStackTrace();
            }
        }
    }

    // フィールド値にアクセスするためのgetterメソッドたち。
    // 命名規則はgetXXX()でXXXにフィールド名が入る。
    public int getHp(){
        return this.hp;
    }
    public int getX(){
        return this.x;
    }
    public int getY(){
        return this.y;
    }

    // ダメージを与える処理（<0対応を忘れずに。）
    public boolean damage(int damage){
        this.hp -= damage;
        if(this.hp <= 0){
            this.hp = 0;
            return true;
        }
        return false;
    }
}
```



```
// 外部からHPに応じた画像ファイルを取得するためのメソッド
// staticフィールドにアクセスするのでthisではなくクラス名.
public Image getImage(){
    if(this.hp < 3)
        return Hatoyama.imgs[3];
    else if(this.hp < 5)
        return Hatoyama.imgs[2];
    else if(this.hp < 8)
        return Hatoyama.imgs[1];
    else
        return Hatoyama.imgs[0];
}
```

## 課題 3：解答例 (GamePanel.java)

```
public class GamePanel extends JPanel implements MouseListener{
    // 全ハトヤマを管理するArrayList
    private ArrayList<Hatoyama> enemies =
        new ArrayList<Hatoyama>();

    // 面倒なのでmain()をGamePanel内に書いてしまうことにする.
    public static void main(String[] args){
        // JFrameを作成してGamePanelをaddして表示する.
        JFrame frame = new JFrame();
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize(500, 500);
        GamePanel panel = new GamePanel(frame.getSize());
        frame.add(panel);
        frame.setVisible(true);

        panel.init();
    }

    // コンストラクタではマウスリスナを登録だけする.
    public GamePanel(Dimension d){
        this.addMouseListener(this);
    }

    // こっそりmain()の末尾に書いたGamePanelの初期化处理
    // ハトヤマを10匹初期値として生成する.
    // これをコンストラクタに書くと、必ず座標0,0に表示されてしまう.
    // これはPanel生成時点ではPanelサイズが確定されておらず、
    // 親のJFrameがVisibleされるまで決定されないためである.
    public void init(){
        for(int i=0;i<10;i++){
            this.generateHatoyama();
        }
        repaint();
    }
}
```

```
// ランダムな位置（画面内）にハトヤマを1匹生成する。
private void generateHatoyama(){
    int x = (int)(Math.random()*(this.getWidth()-100));
    int y = (int)(Math.random()*(this.getHeight()-100));
    // ArrayListに登録しておくことでpaintComponentが処理
    this.enemies.add(new Hatoyama(10, x, y));
}

@Override
public void paintComponent(Graphics g){
    g.setColor(Color.WHITE);
    g.fillRect(0, 0, this.getWidth(), this.getHeight());
    // 色を黒にしておかないとHPが黒くならない。
    g.setColor(Color.BLACK);
    // 全てのハトヤマとそのHPを当該座標に表示する。
    for(Hatoyama h : this.enemies){
        g.drawImage(h.getImage(), h.getX(), h.getY(), this);
        g.drawString(""+h.getHp(), h.getX() + 50, h.getY());
    }
}

@Override
public void mouseReleased(MouseEvent e) {
    // 削除対象を記憶しておくためのArrayList
    ArrayList<Hatoyama> remove =
        new ArrayList<Hatoyama>();
    // 全ハトヤマを探索し座標がヒットしていたらダメージを与える
    boolean flag = false;
    for(Hatoyama h : this.enemies){
        // 衝突判定（マウスX座標がハトヤマX座標～ハトヤマ
        // widthの間にあり、Y座標も衝突の場合を検出する。）
        if(e.getX() >= h.getX() &&
            e.getX() <= h.getX() +
            h.getImage().getWidth(this)){
            if(e.getY() >= h.getY() &&
                e.getY() <= h.getY() +
                h.getImage().getHeight(this)){
                flag = true;
                if(h.damage(1)){
                    // 倒した場合ArrayListから削除しな
                    // ければならないので記憶しておく。
                    // この場で削除すると拡張for文で整合
                    // 性が取れなくなり例外が発生する。
                    remove.add(h);
                }
            }
        }
    }
    // 記憶しておいた倒したハトヤマをまとめて削除する。
    this.enemies.removeAll(remove);
}
```

```
        // クリックに失敗していた場合
        if(!flag){
            // Hatoyamaを増やす
            this.generateHatoyama();
        }
        repaint();
    }

    @Override
    public void mouseClicked(MouseEvent e) {}
    @Override
    public void mousePressed(MouseEvent e) {}
    @Override
    public void mouseEntered(MouseEvent e) {}
    @Override
    public void mouseExited(MouseEvent e) {}
}
```