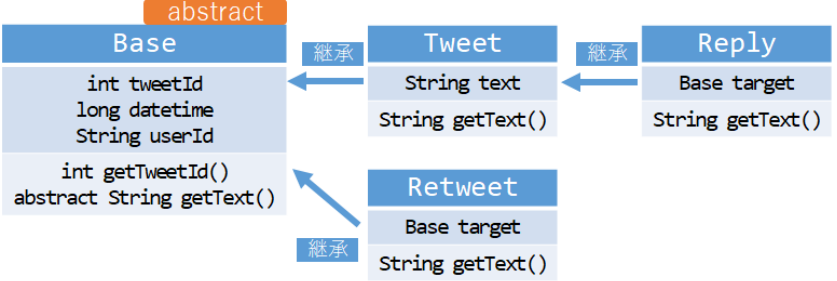


第四回 演習問題（ポリモーフィズム）

諸注意

- 「Base.java」, 「Tweet.java」, 「Reply.java」, 「Retweet.java」, 「TweetManager.java」, 「Client.java」を Web から提出する（6 ファイルを提出する場合、6 回に分けて提出処理を行う）。
- コピー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かない。

課題 1

	問題設定	<p>Twitter は利用者らがつぶやき (Tweet) を投稿するアプリケーションである。Tweet に対して返信 (Reply) を行ったり、再投稿 (Retweet) を行ったりすることができる。</p> <p>Twitter 風アプリケーションを作成するため、次の手順に従って開発を進めよ。基本的にフィールドは”継承されうるクラスの場合子クラスからもアクセスできるように”し、”継承されない場合他のクラスからアクセスできないように”隠蔽せよ。メソッドは他のパッケージからでもアクセスできるようにせよ。オーバーライドのタイミングでは@Override 注釈を忘れないように記述すること。</p>
1	課題 1 で作成するクラス	<p>Base: 以下のオブジェクトの元となるクラス Tweet: 1 つの Tweet を扱うクラス Reply: 1 つの Reply を扱うクラス Retweet: 1 つの Retweet を扱うクラス</p>
	継承関係	 <pre> classDiagram class Base { <<abstract>> +int tweetId +long datetime +String userId +int getTweetId() +abstract String getText() } class Tweet { +String text +String getText() } class Reply { +Base target +String getText() } class Retweet { +Base target +String getText() } Base < -- Tweet Base < -- Retweet Tweet < -- Reply Retweet < -- Retweet </pre> <p>図 1. クラス間の継承関係</p>

ヒント 1：本日の日時を取得するメソッド（課題 1-1 で使用）

本日の日時 (2018/04/11 12:34:56) を 20180401123456 のような long 型で取得するには、以下のメソッドをコピペして使うと良い。内容については将来的に説明する予定である。import 文はファイルの先頭に記述すること。

```

import java.text.SimpleDateFormat;
import java.util.Calendar;

public static long getNow(){
    //Calendarクラスのオブジェクトを生成する
    Calendar c1 = Calendar.getInstance();
    //SimpleDateFormatクラスでフォーマットパターンを設定する
    SimpleDateFormat sdf = new SimpleDateFormat("yyyyMMddhhmmss");
    return Long.parseLong(sdf.format(c1.getTime()));
}

```

1-1	クラス名	Base	備考	抽象クラス
	継承	なし	実装	なし
	フィールド tweetId: Tweet を一意に識別するための ID (int 型) datetime: yyyyMMddhhmmss 形式で日時を示す (long 型) userId: User を一意に識別するための ID (String 型)			
	メソッド int getTweetId(): 自身の tweetId を返す getter() メソッド. String getText(): Tweet の内容を String で返す. 抽象メソッドとする.			

1-1：解答例

```

import java.text.SimpleDateFormat;
import java.util.Calendar;

// Tweetの基礎を扱う抽象クラス
public abstract class Base {
    protected int tweetId = 0;           // Tweet自体のID
    protected long datetime = 0;         // yyyyMMddhhmmss形式とする
    protected String userId = "";        // TweetしたユーザのID

    // 全てのフィールドを初期化する
    public Base(int id, String uid){
        this.tweetId = id;
        this.userId = uid;
        this.datetime = getNow();        // 生成時に自動で決定
    }

    public int getTweetId(){
        return this.tweetId;
    }

    // Tweet内容を返すメソッド (この段階では抽象メソッド)
    public abstract String getText();

    public static long getNow(){
        //Calendarクラスのオブジェクトを生成する
        Calendar c1 = Calendar.getInstance();
        //SimpleDateFormatクラスでフォーマットパターンを設定する
        SimpleDateFormat sdf =
            new SimpleDateFormat("yyyyMMddhhmmss");
        return Long.parseLong(sdf.format(c1.getTime()));
    }
}

```

1-2

クラス名	Tweet	備考	
継承	Base	実装	なし
フィールド text: Tweet のつぶやき内容 (String 型)			
コンストラクタ 引数 : tweetId, userId, text 処理 : 上記 3 引数でフィールドを初期化する。 tweetId, userId, datetime は親クラスの コンストラクタを用いて初期化する。			
メソッド String getText(): 親クラスの getText() を実装する。 <xxx>は変数を意味し、戻り値の出力形式は次の通りとする。 ただし、<>は表示しない。各単語の間は半角スペース 1 つである。@と <userId>は隣接する。 <text> from @<userId> [<datetime>] 例: 初投稿だよ from @t-hase [20180501101010]			
テスト例 (Main.java の main メソッドにて実行する)			
Tweet t1 = new Tweet(0, "t-hase", "初Tweet☆☆"); Tweet t2 = new Tweet(1, "t-hase", "誰もいませんか?"); Tweet t3 = new Tweet(2, "t-hase", "さみしい"); System.out.println(t1 .getText()); System.out.println(t2 .getText()); System.out.println(t3 .getText());			
テスト出力例 初Tweet☆☆ from @t-hase [20180427045831] 誰もいませんか? from @t-hase [20180427045831] さみしい from @t-hase [20180427045831]			

1-2：解答例

```
// 1Tweetを扱うクラス
public class Tweet extends Base{
    protected String text = ""; // Tweetの内容
    public Tweet(int id, String uid, String text){
        // なるべく親クラスのコンストラクタを利用する
        super(id, uid);
        this.text = text;
    }

    // Tweet内容を返すメソッド
    @Override
    public String getText(){
        // <xxx>は変数を意味し, 戻り値の出力形式は次の通りとする.
        // ただし, <>は表示しない. 各単語の間は半角スペース1つである.
        // @と<userId>は隣接する.
        // [text] from @[userId] [[datetime]]<改行>
        return this.text + " from @" + this.userId
            + " [" + this.datetime + "];"
    }
}
```

1-3	クラス名	Reply	備考	
	継承	Tweet	実装	なし
	フィールド target: Reply 対象の Base インスタンス			
	コンストラクタ 引数 : tweetId, userId, text, target 処理 : 上記 4 引数でフィールドを初期化する. tweetId, userId, text, datetime は親クラスの コンストラクタを用いて初期化する.			
	メソッド String getText(): 親クラスの getText() を再実装する. Reply は先頭に@と相手の ID を隣接表示する以外は普通の Tweet と同じなので, Tweet 部は親クラスのメソッドを利用して表示を行うこと. @<target の userId> <text> from @<userId> [<datetime>] 例: @t-hase 初投稿おつ from @s-fuku [20180501101500]			
	ヒント super() や super.method() は親クラスのコンストラクタやメソッドを呼び出す. 親の親クラスのコンストラクタやメソッドではない.			
	テスト例 (Main.java の main メソッドで, 1-2 のテストに続けて書く) Reply rp1 = new Reply(3, "s-fuku", "いますよー", t3); System.out.println(rp1.getText()); テスト出力例 @t-hase いますよー from @s-fuku [20180427050127]			

1-3: 解答例

```
// 1Tweetの一種として, Reply (返信) を扱うクラス
public class Reply extends Tweet {
    // Reply対象のTweet (どのTweetにReplyしたのか)
    // Base型で持つことで, Tweet, Reply, Retweet全てに対応できる.
    protected Base target;

    public Reply(int id, String uid, String text, Base target) {
        // なるべく親クラスのコンストラクタを利用する
        super(id, uid, text);
        this.target = target;
    }

    // Reply内容を返すメソッド
    @Override
    public String getText(){
        // Replyは先頭に@と相手のIDを隣接表示する以外は普通のTweetと
        // 同じなので, Tweet部は親クラスのメソッドを利用する.
        // @<targetのuserId> <text> from @<userId> [<datetime>]
        return "@" + this.target.userId + " " + super.getText();
        // Reply対象のuserIdなので<target.userId>とし, 自身のTweet
        // 表示には親クラス (Tweet) のgetText()を流用する点が重要.
        // <target.userId>はprotectedなのでフィールドアクセス可能.
    }
}
```

1-4	クラス名	Retweet	備考	
	継承	Base	実装	なし
	フィールド target: Reply 対象の Base インスタンス			
	コンストラクタ 引数 : tweetId, userId, target 処理 : 上記 3 引数でフィールドを初期化する. tweetId, userId, datetime は親クラスの コンストラクタを用いて初期化する.			
	メソッド String getText(): 親クラスの getText() を再実装する. Retweet は先頭に RT と表示し Retweet 先の内容をダブルクォーテーションで囲い, その後で from @<userId> [<datetime>] だけ表示する. 例: RT "@t-hase 初投稿おつ from @s-fuku [20180501101500]" from @t-hase [20180501102000]			
	ヒント ダブルクォーテーションを表示するには「¥」 と半角で書くと良い.			
	テスト例 (Main.java の main メソッドで, 1-3 のテストに続けて書く) Retweet rt1 = new Retweet(4, "t-hase", rp1); System.out.println(rt1.getText()); テスト出力例 RT "@t-hase いますよー from @s-fuku [20180427050517]" from @t-hase [20180427050517]			

1-4：解答例

```
// 1Tweetの一種として, Retweet (再投稿) を扱うクラス
// 誰かのTweetをお気に入りとして再投稿することができる機能である.
public class Retweet extends Base{
    protected Base target;

    public Retweet(int id, String uid, Base target) {
        super(id, uid);
        this.target = target;
    }

    // Retweet内容を返すメソッド
    // Retweetは先頭にRTと表示しRetweet先の内容をダブルクォーテーション
    // で囲い, その後でfrom @<userId> [<datetime>]だけ表示する.
    // RT "[元Tweetの内容]" from @[自分のuserId] [[自分のdatetime]]
    @Override
    public String getText(){
        return "RT ¥" + this.target.getText() + "¥" from @"
            + this.userId + " [" + this.datetime + "];
        // Retweet対象の本文を表示するのにtarget.getText()とする.
        // 表示形式はtargetの中身の型にお任せし内容を取得可能
        // targetはBase型なので少なくともgetText()を実装している.
    }
}
```

課題 2

2	問題設定	<p>Twitter 風アプリケーションの基礎データ Tweet 周りの開発を行ったが、main メソッドから利用するには面倒が多い。Tweet を一元管理して取り扱うクラス TweetManager を開発し、Twitter の利用をより簡単にしよう。</p> <p>TweetManager はシステム全体ですべての Tweet を一元管理するため、インスタンス化を禁止するとともに、全てのフィールドとメソッドを static とする。インスタンス化を禁止するには (abstract にはせずに)、コンストラクタを private で宣言するとよい。フィールドは定数のみ公開する。</p>		
	クラス名	TweetManager	備考	インスタンス化を禁止
	継承	なし	実装	なし
	フィールド	<p>MAX_SIZE = 1000: Tweet の上限サイズ (int 型の定数) logs: これまで投稿された全 Tweet を管理する配列 (Base 型配列) nowTweetId: これまで発行された tweetId の最終値 (int 型)</p>		
	コンストラクタ	<p>引数, 処理: なし 備考: private で宣言する (インスタンス化を禁止するため)</p>		
	メソッド	<p>add(): 引数で送られてきた新たな Tweet を保存するメソッド 引数: 保存する Tweet インスタンス (Base 型) 戻り値: なし 処理: logs を検索し null の場合, その要素に引数を格納する。 logs に空きがない場合, 格納せずに以下をコンソール出力する。 上限を超えたため保存されませんでした</p> <p>showAllTweets(): 保存された全ツイートをコンソール出力するメソッド 引数, 戻り値: なし 処理: logs の null 以外の Tweet 全てを順に表示する。 各 Tweet は改行で区切ることとする。</p> <p>getTweetById(): 該当の tweetId を持つ Tweet を検索して返すメソッド 引数: 検索対象の tweetId (int 型) 戻り値: 引数と同じ tweetId を持つ Tweet (Base 型)。 該当しない場合は null を返す。</p> <p>getNewId(): まだ投稿されていない tweetId を返すメソッド 引数: なし 戻り値: まだ投稿されていない tweetId (int 型) 処理: これまで発行された tweetId の最終値+1 を返せば良い。</p>		
	テスト例	<pre>TweetManager.add(new Tweet(TweetManager.getNewId(), "t-hase", "さみしい")); Base t = TweetManager.getTweetById(1); Reply rp = new Reply(TweetManager.getNewId(), "s-fuku", "いますよー", t); TweetManager.add(rp); Retweet rt = new Retweet(TweetManager.getNewId(), "t-hase", rp); TweetManager.add(rt); TweetManager.showAllTweets();</pre>		
	テスト出力例	<pre>さみしい from @t-hase [20180427104833] @t-hase いますよー from @s-fuku [20180427104833] RT "@t-hase いますよー from @s-fuku [20180427104833]" from @t-hase [20180427104833]</pre>		

2：解答例

```
// 全Tweetを管理する敏腕マネージャー
// インスタンス化せずにすべてstaticで管理される.
// すなわち本クラスで定義されたフィールドは、本システムで唯一であり、
// メソッドはインスタンス化せずに利用可能である.
public class TweetManager {
    public static final int MAX_SIZE = 1000;
    private static Base[] logs = new Base[MAX_SIZE];
    private static int nowTweetId = 0;

    // コンストラクタをprivateにすることでnewすることを禁止する.
    private TweetManager(){}

    // 全Tweetを管理するlogsにBase tを格納する
    public static void add(Base t){
        int i = 0;
        // logsを検索していき初めてnullだった場所にTweetを格納する.
        // 拡張for文で実装していた人がいたがnullである場所を見つけて
        // そこに格納を行うため、この場合拡張for文は相性が悪い.
        // 拡張for文は参照のみの場合に使うのが無難だろう.
        for(i=0; i<MAX_SIZE; i++){
            // MAX_SIZEのところはlogs.lengthでもよい.
            if(logs[i] == null){
                logs[i] = t;
                break;
            }
        }
        if(i==MAX_SIZE){
            // 上限を超えてしまった場合
            System.out.println("上限を超えたため
                                保存されませんでした. ");
        }
        // tweetIdとlogs[i]のiは別物なので注意すること.
    }

    // 全Tweetを一括でコンソール出力するメソッド
    public static void showAllTweets(){
        // logsを検索していき、nullでない場合は表示する.
        // 今回は参照だけなので拡張for文を用いても問題ない.
        for(int i=0; i< MAX_SIZE; i++){
            if(logs[i] != null){
                System.out.println(logs[i].getText());
            }
        }
    }
}
```



```
// 全Tweetから該当のtweetIdを持つTweetを取得するメソッド
public static Base getTweetById(int tid){
    // logsを検索していき, nullでない場合はIdチェックを行う.
    // 今回は参照だけなので拡張for文を用いても問題ない.
    for(int i=0; i < MAX_SIZE; i++){
        if(logs[i] != null){
            if(logs[i].getTweetId() == tid){
                return logs[i];
            }
        }
    }
    return null;
}

// まだ投稿されていない次のtweetIdを返す.
// 返したtweetIdは使われたものとする.
public static int getNewId(){
    nowTweetId++;
    return nowTweetId;
}
}
```

課題 3

3	問題設定	<p>Twitter を使うときには一度ユーザ登録を行っておくと煩雑な処理なく簡単に Tweet 等ができる方がよい。しかし、現状は課題 2 のテスト例のように ID を新規取得したり、投稿者の ID を入力して Tweet を生成したりする必要がある。これを簡略化するため、投稿者ごとの Client を開発しよう。</p> <p>Client はインスタンス生成時に投稿者となる利用者の ID を登録しておき、以降の Tweet を簡略化する。フィールドは基本的にすべて隠蔽し、メソッドのみ公開せよ。</p>		
	クラス名	Client	備考	
	継承	なし	実装	なし
	フィールド	userId: 投稿者の ID (String 型)		
	コンストラクタ	引数: userId 処理: 全てのフィールドを初期化する		
	メソッド	tweet(): 引数の内容で Tweet を行うメソッド 引数 : つぶやきの内容 text (String 型) 戻り値 : 生成・保存処理を行った後のインスタンス (Tweet 型) 処理 : Tweet 生成から TweetManager を用いた保存までを行う。 retweet(): 引数の Tweet インスタンスを Retweet するメソッド 引数 : Retweet 対象のインスタンス (Base 型) 戻り値 : 生成・保存処理を行った後のインスタンス (Retweet 型) 処理 : Retweet 生成から TweetManager を用いた保存までを行う。 reply(): 引数の Tweet インスタンスに対して Reply するメソッド 引数 : 1. Reply のつぶやき内容 text (String 型) 2. Reply 対象のインスタンス (Base 型) 戻り値 : 生成・保存処理を行った後のインスタンス (Reply 型) 処理 : Reply 生成から TweetManager を用いた保存までを行う。		
	テスト例	<pre>Client hase = new Client("t-hase"); Client fuku = new Client("s-fuku"); hase.tweet("初Tweet☆☆"); hase.tweet("誰もいませんか?"); Base t1 = hase.tweet("さみしい"); Base t2 = fuku.reply("いますよー", t1); hase.retweet(t2); TweetManager.showAllTweets();</pre>		
	テスト出力例	<pre>初Tweet☆☆ from @t-hase [20180427112256] 誰もいませんか? from @t-hase [20180427112256] さみしい from @t-hase [20180427112256] @t-hase いますよー from @s-fuku [20180427112256] RT "@t-hase いますよー from @s-fuku [20180427112256]" from @t-hase [20180427112256]</pre>		

3: 解答例

```
// Tweetを生成するクライアント
public class Client {
    private String usetId = "";

    public Client(String uid){
        this.ussetId = uid;
    }

    // Tweetを行うメソッド
    public Tweet tweet(String text){
        Tweet t = new Tweet(TweetManager.getNewId(),
                             this.ussetId, text);

        TweetManager.add(t);
        return t;
    }

    // Retweetを行うメソッド
    public Base retweet(Base target){
        Retweet rt = new Retweet(TweetManager.getNewId(),
                                   this.ussetId, target);

        TweetManager.add(rt);
        return rt;
    }

    // Replyを行うメソッド
    public Base reply(String text, Base target){
        Reply rp = new Reply(TweetManager.getNewId(),
                              this.ussetId, text, target);

        TweetManager.add(rp);
        return rp;
    }
}
```