第二回　演習問題（クラス，継承，カプセル化の復習）

諸注意

- 「Item.java」，「Food.java」，「FoodEx.java」を Web から提出する（３ファイルを提出する場合，３回に分けて提出処理を行う）．
- 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．例えば課題１の purchase() 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	商品管理システムを開発したい．まずは商品を取り扱うクラス Item を開発せよ．ただしフィールドは全て隠蔽し，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．
	フィールド	name: 商品名(文字列型) price: 単価(数値型) stock: 在庫数(数値型)
	コンストラクタ	1．全てのフィールドの初期値を設定する．
	メソッド	int getStock(): 在庫数を確認するメソッド 　引数　：なし 　戻り値：現在の在庫数 int getPrice(): 単価を確認するメソッド 　引数　：なし 　戻り値：商品の単価 int add(int num): 在庫を補充するメソッド 　引数　：補充する数 　戻り値：補充後の在庫数 　処理　：在庫を引数の数だけ補充する int purchase(int num): 商品を購入するメソッド 　引数　：購入する数 　戻り値：購入に必要な金額 　処理　：購入数を在庫から減ずる．ただし減ぜないときは在庫数を変えずに戻り値-1 を返す．

	テスト例	Main.java の main メソッドにて
		`Item cup = new Item("カップ", 100, 20);`
		`System.out.println(cup.getStock());`
		`System.out.println(cup.getPrice());`
		`cup.add(10);`
		`System.out.println(cup.getStock());`
		`System.out.println(cup.purchase(15));`
		`System.out.println(cup.purchase(20));`
		`System.out.println(cup.getStock());`

	出力例	20 100 30 1500 -1 15

１：解答例

```
public class Item {
	// nameはどこでも使用されていないので
	// 「フィールド Item.name の値は使用されていません」
	// というWarningが出ますが，気にしなくて良いです．
	private String name = "";
	private int price = 0;
	private int stock = 0;

	// 初期値をセットするときにthis.を忘れないように注意する．
	public Item(String name, int price, int stock){
		this.name = name;
		this.price = price;
		this.stock = stock;
	}

	// いわゆるgetter()です．
	public int getStock(){
		return this.stock;
	}

	// いわゆるgetter()です．
	public int getPrice(){
		return this.price;
	}

	// 元の在庫数に加算する＝補充する
	public int add(int num){
		this.stock += num;
		return this.stock;
	}

	public int purchase(int num){
		// １．在庫が足りるかどうかを確認する．
		if(num <= this.stock){
			// ２．足りる場合は在庫から販売数を減じて必要価格を返す．
			this.stock -= num;
			return num * this.price;
		}else{
			// ３．不足する場合は-1を返す．
			return -1;
		}
	}
}
```

課題２

２	問題設定	たくさんの商品は Item クラスで管理できているが，食べ物に関してのみ賞味期限を考えねばならず困っている．Item クラスを継承して，食べ物に特化した Food クラスを開発したい．ただしフィールドは全て隠蔽し，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．
	フィールド	limit: 賞味期限（数値型で 20180417 のように管理する）
	コンストラクタ	１．　全てのフィールドの初期値を設定する． ※親クラスのコンストラクタを利用すること．
	メソッド	boolean checkLimit(int limit)： 賞味期限切れの確認メソッド ・当日はセーフとする 引数　：確認する日付 戻り値：賞味期限が切れている(false) 切れていない(true)

	テスト例	Main.java の main メソッドにて
		`Food apple=new Food("りんご",100,10,20180401);`
		`System.out.println(apple.checkLimit(20180331));`
		`System.out.println(apple.checkLimit(20180401));`
		`System.out.println(apple.checkLimit(20180402));`

	出力例	true true false

２：解答例

```
public class Food extends Item{
        private int limit = 0;

        // super()で親クラスのコンストラクタを使うことを忘れずに．
        // (そもそも3変数はprivateなのでコンストラクタ経由でしかセットできない)
        public Food(String name, int price, int stock){
                super(name, price, stock);
        }

        // 上記同様親クラスのコンストラクタを使用し，limitのみ自分で初期化する．
        // 上で定義したコンストラクタを用いてthis(name,price,stock);でも良い．
        public Food(String name, int price, int stock, int limit){
                super(name, price, stock);
                this.limit = limit;
        }

        // 賞味期限チェックの最もシンプルな手法である．
        // this.limit>=limitはbooleanの結果になるためそれをそのまま返している．
        public boolean checkLimit(int limit){
                return this.limit >= limit;
        }
}
```

課題３

3-1	問題設定	Food クラスを開発したが運用上の問題が発覚した．賞味期限 20180101 で 100 個りんごを作成し，賞味期限 20180201 で 50 個りんごを追加したいとする．この時親クラスの add メソッドでは賞味期限の更新に対応できない． 二引数の独自メソッドを作成したとしても賞味期限と在庫数を複数管理することができない．すなわち，20180101 のりんごを 150 個もしくは 20180201 のりんごを 150 個という管理しかできないわけである． Item クラスを継承し，これらの問題に対応したクラス FoodEx を開発せよ．ただしフィールドは全て隠蔽し，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．
	フィールド	stocks: 賞味期限をキーとして在庫数を管理するマップ TreeMap<Integer,Integer>型：TreeMap の使い方はヒント１
	コンストラクタ	１．　name，price の初期値を設定する． ※親クラスのコンストラクタを利用すること． ※stock には 0 を代入しておくこと（FoodEx では stock は使用しない）
	メソッド	int add(int limit, int num)： limit が賞味期限の在庫を num 個補充するメソッド 引数　：limit: 今回補充する在庫の賞味期限 ：num: 今回補充する在庫の数 戻り値：補充後の在庫数（次の getStock() を使うとよい） 処理　：引数に従って stocks に在庫を補充する int getStock()： 在庫数を確認するメソッド • stocks から算出する（stock は使用しない） • 賞味期限は問わない（全てカウントする） • オーバーライドすること • デバッグ用のコンソール出力を残さないように 引数　：なし 戻り値：現在の在庫数

	テスト例	Main.java の main メソッドにて
		`FoodEx f = new FoodEx("りんご", 100);`
		`f.add(20180401, 10);`
		`f.add(20180501, 10);`
		`f.add(20180401, 10);`
		`f.add(20180601, 10);`
		`System.out.println(f.getStock());`

	出力例	40

<table>
<tr><td rowspan="3">3-2</td><td>問題設定</td><td>FoodEx クラスのメソッドをもう少し拡充しよう.</td></tr>
<tr><td>メソッド</td><td>void refresh()：
賞味期限切れの在庫を破棄するメソッド
引数　：なし
戻り値：なし
処理　：メンバ変数 stocks の中から賞味期限切れの在庫数を0で更新する.（本日の日付を取得する方法はヒント2を参考にせよ.）

int purchase(int num)：
商品を購入するメソッド
・先頭で refresh()を呼び出すこと
・オーバーライドすること
引数　：購入する数
戻り値：購入に必要な金額
処理　：在庫数が足りないときには-1 を返し，在庫数は変更しない．在庫数が足りるときは，賞味期限の早い商品から順に在庫数減ずる.</td></tr>
<tr><td>テスト例</td><td><pre>
FoodEx f = new FoodEx("りんご", 100);
f.add(20180401, 10);
f.add(20180501, 10);
f.add(20180401, 10);
f.add(20180601, 10);
System.out.println(f.getStock());

f.refresh();
System.out.println(f.getStock());

System.out.println(f.purchase(5));
System.out.println(f.purchase(20));
System.out.println(f.purchase(10));
System.out.println(f.getStock());
</pre></td></tr>
<tr><td></td><td>出力例</td><td>40
20
500
-1
1000
5</td></tr>
</table>

3：解答例（3-1 と 3-2 をまとめて）

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

public class FoodEx extends Item{
	private TreeMap<Integer, Integer> stocks = new TreeMap<Integer,
Integer>();

	// 今回フィールドのint型stockを使用せずにTreeMap型stocksを
	// 用いることにしたため，とりあえずstockは0で初期化しておく．
	public FoodEx(String name, int price){
		super(name, price, 0);
	}


	public int add(int limit, int num){
		// stocksに同じ日付がキーで登録されていれば在庫数を補充する．
		if(this.stocks.containsKey(limit)){
			int now = this.stocks.get(limit);
			this.stocks.put(limit, num+now);
		}else{
			// 同じ日付がない場合には新たにキーと在庫数を登録する．
			stocks.put(limit, num);
		}

		// 後に定義するgetStock()を流用すると手っ取り早い．
		return this.getStock();
	}

	// ちゃんとオーバーライドであることを明示する．
	@Override
	public int getStock(){
		int sum = 0;
		// 拡張for文で全てのKeyをループさせる．
		for(int key : this.stocks.keySet()){
			// 各Keyに対応する値をgetして合計していく．
			sum += this.stocks.get(key);
		}
		return sum;
	}
```

```
        // 内部処理は複雑になったが，本クラスをインスタンス化して使う側から見る
        // と，数を入力して購入処理を行うということに変わりはないように見える.
        @Override
        public int purchase(int num){
                this.refresh();
                // 初めに戻り値用に合計金額を計算しておく
                int total = num*this.getPrice();

                // 今回の購入数に対して在庫が足りるのかを確認する
                if(this.getStock()>=num){
                        // 拡張for文で全てのKeyをループさせる
                        // TreeMapなので自動でKey昇順でソートされている.
                        for(int key : this.stocks.keySet()){
                                // 賞味期限の早いものから順にアクセスできる
                                // ので，可能な量だけ在庫を減じていく.
                                int s = this.stocks.get(key);
                                if(s < num){
                                        num -= s;
                                        this.stocks.put(key, 0);
                                }else{
                                        s -= num;
                                        num = 0;
                                        this.stocks.put(key, s);
                                        break;
                                }
                        }
                }else{
                        // 在庫が足りない場合は-1を返す
                        return -1;
                }
                return total;
        }

        public void refresh(){
                int today = FoodEx.getToday();
                // 拡張for文で全てのKeyをループさせる.
                for(int key : this.stocks.keySet()){
                        // Key（賞味期限）が本日の日付より前であれば
                        //在庫数を0で上書きする.
                        if(key < today){
                                this.stocks.put(key, 0);
                        }
                }
        }

        // これはヒントに書いたままのものである.
        public static int getToday(){
        //Calendarクラスのオブジェクトを生成する.
        Calendar cl = 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("yyyyMMdd");
        return Integer.parseInt(sdf.format(cl.getTime()));
        }
}
```

ヒント１：TreeMap の使い方

TreeMap は HashMap の様に「キー」と「値」を１：１で保管する Map の一種である．すなわち，20180401 というキーに 20 という在庫数をセットで保管したい場合などに使用する．二分探索を用いて検索するため，キーがソートされている点が特徴である．

```
import java.util.TreeMap; // 先頭に記述する
TreeMap<Integer, Integer> stocks = new TreeMap<Integer, Integer>();
```

でインスタンスを生成する．import 文はファイルの先頭に記述する必要がある．ここで<>でどのような型を取り扱うのかを宣言している．最初の Integer はキーの型，二つ目は値の型である．今回の問題の場合はキーが 20180401 のような日付，値が在庫数となるため両方 Integer（int 型）とした．

キーと値のペアを追加したい場合 put(キー，値)メソッドを使用する．また，キーがすでに登録されているかを確認するには containsKey(キー)を用いる．登録されている際に，キーに対応する値を取得するには get(キー)を用いる．例えば，20180401 が賞味期限の在庫を 20 個補充し，20180401 が登録されているのかを確認し，登録されている場合該当の値を取得するときには以下のように記述する．

```
stocks.put(20180401, 20);
if (stocks.containsKey(20180401)){
    int value = stocks.get(20180401);
}
```

ここで，put()メソッドはもともとあった値を上書きしてしまうため，補充を行う際には何かしら工夫を行う必要がある．キーと値のペアを削除したい場合，本来は remove(キー)を使用するが，今回はキーに該当する値を 0 で上書きすれば良い．

また，登録されているすべてのキーを参照したい場合は拡張 for 文を用いて以下のように記述する．

```
for( int key : stocks.keySet()){
  int value = stocks.get(key);
}
```

ヒント２：本日の日付を取得するメソッド

本日の日付を 20180401 のような int 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 int getToday(){
    //Calendarクラスのオブジェクトを生成する
    Calendar cl = 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("yyyyMMdd");
    return Integer.parseInt(sdf.format(cl.getTime()));
}
```