第 50 回検定（平成 26 年 1 月 19 日）全商情処プロ部門 Java 解説

１級 【７】

解答

(1)	total += quantity （別解　total = total + quantity）
(2)	p = start; p <= end
(3)	rank[scope][q] += 1 （別解 1　rank[scope][q]++ 別解 2　rank[scope][q] = rank[scope][q] + 1）
(4)	obj[m].getCode() < code
(5)	obj[m].calcSales(quantity, kind)

オブジェクト指向を取り入れた難問である。クラス Menu を継承し、サブクラス LunchMenu と SideMenu をインスタンス化している。また、calcSales メソッドはオーバーライドされている。このようなオブジェクト指向独特の内容が問いでも問われており、前回検定より遥かに難しくなっている。

（１）は、LunchMenu や SideMenu のスーパークラスである Menu クラスのインスタンスメソッドの箇所である。この calcSales メソッドは、2 つのサブクラスの super.calcSales…の箇所で呼び出されることとなる。この super.calcSales は else の中で呼び出されていることから解るように、販売種別が 0 の時に呼び出される。この時は単に売上数量計と売上金額を足せばよい、ということが解れば（１）は解けるであろう。また、（１）の上にある if(kind == 0)は無くても同じ動作をする。なぜなら、サブクラスの else から super.calcSales が呼び出されるので、その時点で kind == 0 だからである。これは明示的に if(kind == 0)とすることにより、プログラムを解り易くしている、と思われる。
また、サブクラス内でオーバーライドした calcSales メソッドでは、（１）に該当する処理は、setTotal(getTotal() + quantity)となっている。これは、getTotal()というゲッターで呼び出した値に quantity をプラスし、それを setTotal()というセッターで保存している、というものである。Menu の変数である total のアクセス修飾子は private であり自クラスからしか参照できない。サブクラスからも直接参照できないので、こうやってゲッターやセッター経由で total にアクセスしている、ということである。
ちなみに、quantity とは「量、数量」のことである。

（２）の for は、rank[scope][p] = 1 となっていることから、順位を初期化していることが推測できる。ここで、「わかった！答えは『p = 0; p < SIZE』だ」と飛びついてしまうと、私の様に間違えてしまう。この addRank という **static メソッド**は、プログラムの後半で 3 回連続して呼び出されている。

1回目のaddrank(0, SIZE－1, SCOPE0)は、全体順位を付けている。2回目のaddrank(0, idx, SCOPE1)

は、弁当の分類別順位を付けている。3回目のaddrank(idx + 1, SIZE - 1, SCOPE1)は、サイドメニューの分類別順位を付けている。このように、付ける順位の場所が違うので、（２）で初期化する場所も変える必要がある。そう考えると、引数で受け取っているstartとendを使用することがわかるであろう。

しかし、ここで問題となるのが、2回目と3回目に呼び出すときに使用している変数idxである。このidxは、クラスSellAnalystの真ん中あたりのswitch(type)のcase 1の場合に更新されている。このcase 1の場合とは、100番台（つまり弁当）の場合であり、100番台を処理するたびにidxが更新されることになる。データは商品コードの昇順に記録されているので、最後にcase 1を通り、最後にidxが更新された時、そのidxは100番台の最後のデータ場所を表すこととなる。つまりidxとは100番台最後のデータ位置である。これがidxの意味である。

static メソッド

インスタンス化しなくても呼び出せるメソッド。publicやfinalと混じってしまって区別がつかなくなることがあるので、じっくり理解したい。calcSalesやoutListはインスタンスメソッドでありインスタンス化しなければ使用できず、この static メソッドとは区別する。クラスSellAnalystにはstaticな変数やstaticなメソッドが多数存在する。staticメソッドはインスタンス化しなくても呼び出せる半面、多用すると非オブジェクト指向なプログラムに陥ることが多い。

（３）はifからも解るように順位をつける命令文である。obj[p]とobj[q]の関係を考えるとどちらの順位を足せばよいかわかるであろう。なお、[p]と[q]は似ているので読み解くときや記述する時に注意が必要である。

ちなみに配列objは、Menu[] obj = new Menu[SIZE]と宣言されている。そのMenu型の配列にサブクラスであるLunchMenuやSideMenuのインスタンスの参照を入れている、ということである。このように、インスタンスの参照を配列に入れることができること、また、スーパークラスにはサブクラスの参照を代入することができるということを理解しておく必要がある。

（４）も（３）と同様に古典的な問題である。（４）は二分探索の中央値を上限か下限のどちらにするか決める条件と言うことがすぐに理解できる。codeと比較する配列objの中にある商品コードはgetCodeというゲッターを使用する必要がある。これは（４）の上にあるwhileを見るとある程度推測がつく。しかし、上記でも説明したように、配列 obj にはインスタンス化された LunchMenu やSideMenuの参照が格納されている、ということを理解していなければ解答するのは難しい。

（５）は、二分探索をして結局何をしたいのか、ということを分かる必要がある。二分探索をして一致するコードを探して、結局売上数量計や売上金額等の計算をしたいのである。計算をするためにはcalcSalesメソッドを呼び出す必要がある。そこで、サブクラスLunchMenuやSideMenuのcalcSalesを見てみると、引数としてint quantityとint kindを使用している。これらは（５）の上にある、quantityとkindを渡すことになるので、解答はobj[m].calcSales(quantity, kind)となる。たまたま変数の名前がサブクラスの calcSales で使用している名前と同じである。しかし、サブクラスの

calcSales で使用している quantity と kind は、calcSales 内だけで使用できる変数であり、（５）の上にある quantity と kind とは別物である。変数の名前が同じだからと言っても変数の有効範囲（スコープ）が違っているので、この点も慣れが必要となる。

`private static final int SIZE = 50;　など`

final とは定数を表す。定数とはプログラム上で変わらない数のことである。変数 SIZE はプログラムの途中で SIZE = 40 などと変更することはできない。また、定数は大文字で表現する、ということが Java の一般的なコーディング規約なので、size ではなく SIZE で定義されている。