問11　ブロックのデータのキャッシュ管理（Java）　(H27秋-FE 午後問11)

【解答】

［設問］　a－ウ，b－ア，c－エ，d－オ，e－イ，f－カ，g－ウ

【解説】

ブロックデータへのアクセス管理プログラムの問題である。プログラムでは，java.utilパッケージのListインタフェース，及びクラスArrayListが使われている。また，java.langパッケージのクラスSystemのメソッドarraycopyも使われており，「Javaプログラムで使用するAPIの説明」で仕様を確認しながら解答する必要がある。

プログラムでは，ブロックアクセッサを表すクラスBlockAccessor，ブロックデバイスを表すクラスBlockDeviceが定義されている。また，キャッシュ機能を実現するためのクラスとして，抽象クラスCacheと，クラスCacheのサブクラスである，抽象クラスListBasedCacheが定義されている。さらに，クラスListBasedCacheには，入れ子クラスとして，Entry，Fifo，Lruの三つが定義されている。

午後のプログラムの問題は，複数の設問から構成されることが多いが，本問は設問が一つしかない。またメソッドmainをもつテスト用のクラスもない。設問は，全てプログラムコードの穴埋めになっている。空欄は，プログラムの説明とプログラムコードの内容をよく読み，インタフェースListの機能を利用した，キャッシュの機能の仕組みを把握することで解答できる。またFIFO，LRUについての知識が問われる。

［設問］

・空欄a：クラスBlockDeviceのメソッドgetBlockSizeの処理である。メソッドgetBlockSizeの処理内容については〔プログラムの説明〕(2)②に「ブロックのサイズをバイト数で返す」とある。空欄aは，メソッドgetBlockSizeの戻り値となっており，空欄aを含む文の値が「ブロックのサイズをバイト数」で表した値になればよい。空欄aの右側を見ると，「.length」となっている。「.length」の構文を利用できるのは，「.length」の左が配列を参照する変数の場合である。メソッドgetBlockSize内では，配列の変数を宣言していないので，フィールドで宣言された配列を参照することが分かる。クラスBlockDeviceのフィールドは，byte型の2次元配列blocksだけであるので，フィールドblocksを利用した構文であればよいことが分かる。フィールドblocksは2次元配列であるので，どちらの次元が個々のブロックを表しているか判別する必要がある。〔プログラムの説明〕には，どちらの次元が個々のブロックを表しているかの説明はない。そこで，メソッドreadBlockのプログラムから読み取る。メソッドreadBlockについては，〔プログラムの説明〕(2)③に記述がある。引数がインデックス値となっているので，引数を配列の要素番号（添字）として利用している次元が，ブロックのインデックスを表す次元であり，他方が個々のブロックを表す次元であることが分かる。プログラムを見ると，メソッドreadBlockの1行目で「byte[] block = blocks[index];」と記述されている。このことから，フィールドblocksの1次元目は，ブロックのインデックスを表しており，2次元目は，個々のブロックを表していることが分かる。2次元配列において，1次元目のある要素が指す，2次元目の配列の要素数を取得するためには「blocks.[要素番号].length」のように，1次元目の要素番号を指定して，「.length」構文を記述する。Javaの2次元配列では，要素数の異なる配列を2次元の配列とすることができるが，この問題のプログラムにおいては，フィールドblocksの宣言以外で，2次元の要素数を変更するような処理はない。このため，1次元目の要素番号は0から99のどれを指定しても，2次元目の配列の要素数は同じ値となる。選択肢でフィールドblocksの1次元の要素数を指定している構文は「blocks[0]」である。したがって，空欄aには（ウ）が入る。

・空欄b：クラスCacheのメソッドcreateCacheの宣言構文の，戻り値の型である。〔プログラムの説明〕(3)②に「引数で指定されたキャッシュの管理方針と一致する実装クラスのインスタンスを生成して返す」とある。「実装クラス」が具体的に何であるかが明記されていないので，プログラムから読み取る。戻り値の型を知りたいので，メソッドcreateCacheのreturn文の部分を見てみる。メソッドcreateCache内の3行目に「return new ListBasedCache.Fifo();」，5行目に「return new ListBasedCache.Lru();」とある。この二つに共通する型が戻り値の型となればよい。〔プログラムの説明〕には，この二つのクラスの継承関係に関する説明はない。また，プログラムを見ても，継承部分は空欄fになっている。そこで，プログラム中でメソッドcreateCacheを呼び出している箇所で，何型の変数に代入しているか見てみる。クラスBlockAccessorのコンストラクタの2行目で，Cache型のフィールドcacheに，メソッドcreateCacheの戻り値を代入している。このことから，戻り値の型はCache型であればよいことが分かる。解答群の中で，Cache型として扱えるのは，「Cache」だけである。したがって，（ア）が入る。

・空欄c：クラスListBasedCacheのメソッドgetCachedBlockDataの処理である。空欄cは，拡張for文の繰返し処理内のif文の条件式の比較演算子である。メソッドgetCashedBlockDataの処理については，〔プログラムの説明〕(3)③に「引数で指定されたインデックス値のブロックデータがキャッシュされていれば，それを返す」とある。この処理をどのように実装しているか理解するためプログラムを見ていく。この問題の説明とプログラムから読み取ると，クラスListBasedCacheのフィールドentriesがキャッシュの役割を果たしていることが分かる。キャッシュに格納される「キャッシュエントリ」は，クラスListBasedCacheの入れ子クラスであるEntryのインスタンスに「ブロックデータとそのインデックス値の対」として格納される。これを踏まえて，空欄cを含むif文を内包する拡張for文について見てみる。拡張for文では，コロン記号の右側に，走査対象となる，配列やコレクションを参照する変数，又はフィールドを記述する。一方，コロン記号の左側には，走査対象の各要素を参照する変数の型と変数名を記述する。これらを整理すると，空欄cを内包する拡張for文は，キャッシュ（entries）を走査して，キャッシュに登録されている各要素であるキャッシュエントリ（インデックス値とブロックデータの対）を参照変数entryで参照していることが分かる。

次に空欄cを含むif文の条件式がtrueの場合の処理を見ると，メソッドhitを呼び出した後，参照変数entryのメソッドgetBlockDataを呼び出し，その戻り値をメソッドgetCachedBolockDataの戻り値としている。この内容と拡張for文の処理を合わせて考えると，空欄cを含むif文は，引数indexとキャッシュエントリのインデックスが同じ値の場合にtrueになる条件式であればよい。空欄cの右辺は引数のindexであり，メソッドの説明から，キャッシュされているか確認するブロックのインデックス値である。一方，左辺は，「entry.getIndex()」となっておりどちらもint型なので，比較演算子として「==」を使うことで等しいかどうか確認できる。したがって，（エ）が入る。

・空欄d：空欄dは，クラスListBasedCacheのメソッドcacheBlockDataの処理であり，if文の条件式である。メソッドcacheBlockDataについては〔プログラムの説明〕(3)④に「キャッシュできるブロックデータ数が上限に達している場合は，管理方針に従ってキャッシュされているブロックデータを削除してから，指定されたブロックデータをキャッシュする」とある。空欄dの1行下に，entries.removeというブロックデータの削除を行う記述があることから，「キャッシュできるブロックデータ数が上限に達している場合」に該当するif文であることが分かる。キャッシュできる上限については，〔プログラムの説明〕(4)に「キャッシュできるブロックデータ数の上限値は，フィールドCACHE_SIZEで表す」とある。現在キャッシュされているブロックデータ数は，フィールドentriesに対してメソッドsizeを呼び出せば取得できる。なお，メソッドsizeについてはAPIの説明で仕様を確認できる。以上のことから，空欄dの条件式は「entries.size() == CACHE_SIZE」であればよい。したがって，（オ）が入る。

・空欄e：空欄eは，フィールドentriesに対するメソッドremoveの呼出しの引数である。削除されるキャッシュエントリは〔プログラムの説明〕(3)④「管理方針に従って」削除されなければならない。メソッドcacheBlockDataについての実装は，クラスListBasedCacheにしかないので，ここでの処理は，各管理方針を満たす処理内容となっていることが分かる。そこで，管理のアルゴリズムが単純なFIFOの場合で考えてみる。FIFOでは，最初にキャッシュに格納されたキャッシュエントリが（First In），最初に削除対象になる（First Out）。この問題のプログラムでは，メソッドchacheBlockDataでキャッシュにキャッシュエントリを格納している。その際，インタフェースListのメソッドaddを呼び出し，第1引数で0を指定しているので，リストの要素番号0の要素に追加していることになる。インタフェースListのメソッドaddの仕様は，APIの説明で確認できる。APIの説明には「指定された位置及びその後に既に要素がある場合は，それらの要素をシフトする」とある。この説明を具体的に考えると，仮に要素番号0にキャッシュエントリAが存在する状態で，メソッドaddを呼び出し，さらにキャッシュエントリBを要素番号0に追加すると，キャッシュエントリAの格納位置は，要素番号1に移動されることになる。つまり，最初に追加したキャッシュエントリAは，リストのサイズ2から1減算した要素番号1の位置に格納されることになる。FIFOでは最初に追加されたキャッシュエントリが，最初に削除されるので，キャッシュサイズの最大値の上限に達した状態で，削除されるのは，キャッシュサイズの最大値から1減算した要素番号の位置に格納されたキャッシュであればよい。これをプログラム中のフィールドCACHE_SIZEを利用して式で表すと「CACHE_SIZE - 1」となる。したがって，空欄eには（イ）が入る。

・空欄f：空欄fは，クラスFifo，及びクラスLruのクラス宣言部のスーパクラスを指定する部分である。〔プログラムの説明〕には，継承関係に関する記述はない。そこで，プログラムから，どのクラスがスーパクラスであるべきか読み取る。両方のクラスで，実装されているのは，メソッドhitである。メソッドhitは，クラスListBasedCacheで抽象メソッドとして定義されている。このため，クラスListBasedCacheを継承したサブクラスが，具象クラス（abstract修飾子がないクラス）である場合，メソッドhitを必ず実装する必要がある。このことから，クラスFifo，及びクラスLruのスーパクラスは，クラスListBasedCacheであればよいと分かる。したがって，（カ）が入る。

・空欄g：空欄gは，クラスLruのメソッドhitの処理である。空欄gは，フィールドentriesに対するメソッドremoveの呼出しの引数である。メソッドhitについては，〔プログラムの説明〕(4)②に記述がある。さらに，クラスLruのメソッドhitについては，〔プログラムの説明〕(7)①に記述がある。クラスLruで処理されるキャッシュの管理は，LRUのアルゴリズムで行われる。LRUでは，キャッシュの上限に達している場合，最も長い間使われていないキャッシュエントリが削除対象になる。このプログラムでは，空欄eで見たように，キャッシュの要素のCACHE_SIZE - 1の要素番号にあるキャッシュエントリが削除される。このアルゴリズムで，LRUを実現するためには，最近参照されたキャッシュエントリをキャッシュの要素番号の0の位置に移動させればよい。このことを踏まえてプログラムを見ていく。メソッドhitが呼び出されるのは，メソッドgetCachedBlockDataで，キャッシュにキャッシュエントリがあった場合である。言い換えると，キャッシュにあったキャッシュエントリが参照された場合である。また，メソッドgetCachedBlockDataで，メソッドhitに渡される引数は，参照されたキャッシュエントリである。参照されたキャッシュエントリは，最近参照されたキャッシュエントリとなるため，空欄gを含む行で，キャッシュエントリを削除し，空欄gを含む行の1行下で，キャッシュの要素番号0の位置に追加することで，キャッシュの要素番号の0の位置に移動することを実現している。つまり，空欄gのメソッドremoveでは，引数entryのキャッシュエントリを削除すればよい。インタフェースListのAPIの説明には，引数の異なる二つのメソッドremoveの説明がある。Javaにおいて，クラスObjectは，全てのクラスのスーパクラスであり，引数がObject型である場合，全てのクラスから生成されたインスタンスを引数として渡すことができる。ここでは，削除する対象が，引数「entry」が参照するオブジェクトであるため，引数がObject型の方のメソッドremoveの呼出しを利用すればよい。したがって，（ウ）が入る。