Which is the most important?

A) Name should be between 10 to 16 characters inclusive
   \texttt{ruleset = SIMPLE\_CHESS;}

B) Unique within the program
   \texttt{int gameCount;}
   Two local variables in two classes are both called \texttt{gameCount}

C) Accurately describes the entity or purpose
   \texttt{boolean \textcolor{red}{flag} = true;}

D) Correctly follow's convention e.g. Java's camelCase
   \texttt{boolean \textcolor{red}{networkerror} = false;}

What is the problem with this code?

```java
String message = textGen.toHTML(template, textConfig);
List<Player> onlinePlayers = server.getCurrentPlayers();

assert (onlinePlayers != null) && (message != null);
Log("Sending messages to players " + onlinePlayers.length);
for (Player onlinePlayer : onlinePlayers) {
    if ((onlinePlayers != null) && onlinePlayer.notify(message)) {
        sentCount ++;
    }
}
Log("Message sentCount : " + sentCount);
```
Poor choice for a boolean variable name?

A) notFound
B) hasChildren
C) processingComplete
D) partitionClean
Which of the following is **not** a reason the book advocates for creating data structures?

A) to improve program efficiency  
B) to clarify data relationships  
C) to simplify operations on blocks of data  
D) to simplify parameter lists  
E) to reduce maintenance
What is 'debugSQL'?

```java
getBooks(email, debugSQL) {
    user = getUserFromEmail(email);
    libraryBooks = getLibraryBooks(user, debugSQL);
    return libraryBooks;
}

getLibraryBooks(user, debugSQL) {
    Log("Getting library books for user " + user);
    return readDatabase(user, SELECT_BOOKS, debugSQL);
}

readDatabase(user, query, debugSQL) {
    return sqlSELECT(query, params, debugSQL);
}

sqlSelect(query, params, debugSQL) {
    if (debugSQL) {
        ...
    }
```
Which is better?

A. Direct use of Global objects (= public static objects in Java)

App.queue[App.queueSize ++ ] = item;

Later...

if (App.queueSize > 0) {
  Item item = App.queue[App.queueSize ];
}

B. Encapsulate access

App.enqueueItem( item );

Item item = App.tryDequeueItem();
if( item != null ) {

C. Both are fine
Pick the best answer. Global data...

- Damages modularity and maintainability
- Can cause problems with multi-threaded code
- Updates to global data may not be obvious (e.g., embedded in other function calls)
- the initialization order of global data may not be defined or obvious
- or…

All of the above.
Named constants:
Which statement is most **FALSE**?

A) Variable names for named constants should ALL_CAPS with underscores separating the words
B) Can be changed more reliably than magic values
C) 0 and 1 should be the only number literals that should occur in the body of the code
D) Make the code more readable than magic values
E) Should declared in a separate class
static boolean validSignal(int[] sensor, float sensitivity) {
    assert sensor.length != 0;
    int total = 0;
    int validCount = 0;
    for (int i = 0; i < data.length; i++) {
        if (sensor[i] != -999999999) {
            validCount++;
            total += sensor[i];
        }
    }
    return (total / sensitivity) > Math.sqrt(validCount);
What is a POJO?

class Person1 {
    private String name;
    private String email;
    // getters and setters
}

class Person2
    extends JSONWriter
    implements MyDatabaseEntity {
    private String name;
    private String email;

    static Person2 getById(key);
    updateDatabase();
    String toJSON();
}
Types, Naming, and Table-Driven Methods
Scrabble
Scrabble word score

Sum of the letter values

English-language editions of Scrabble contain 100 letter tiles, in the following distribution:

- 2 blank tiles (scoring 0 points)
- 1 point: E ×12, A ×9, I ×9, O ×8, N ×6, R ×6, T ×6, L ×4, S ×4, U ×4.
- 2 points: D ×4, G ×3.
- 3 points: B ×2, C ×2, M ×2, P ×2.
- 4 points: F ×2, H ×2, V ×2, W ×2, Y ×2.
- 5 points: K ×1.
public static int wordScore(String word) {
    int score = 0;
    for (int i = 0 ; i < word.length() ; i++) {
        char letter = word.charAt(i);
        score += letterScore(letter);
    }
    return score;
}
Control-flow based

```java
public static int letterScore(char c) {
    char upperC = Character.toUpperCase(c);
    switch (upperC) {
        case 'A':
        case 'E':
        case 'I':
        case 'L':
        case 'N':
        case 'O':
        case 'R':
        case 'S':
        case 'T':
        case 'U':
            return 1;
        case 'D':
        case 'G':
            return 2;
        case 'B':
        case 'C':
        case 'M':
        case 'P':
            return 3;
        case 'F':
        case 'H':
        case 'V':
        case 'W':
        case 'Y':
            return 4;
        case 'K':
            return 5;
        case 'J':
        case 'X':
            return 8;
        case 'Q':
        case 'Z':
            return 10;
        default:
            // handle error
            return 0;
        }
    // should never reach here
    return 0;
}
```
Table-based Solution

private static final int[] scoresByChar =
    { /* A */ 1, /* B */ 3, /* C */ 3, /* D */ 2, /* E */ 1,
      /* F */ 4, /* G */ 2, /* H */ 4, /* I */ 1, /* J */ 8,
      /* K */ 5, /* L */ 1, /* M */ 3, /* N */ 1, /* O */ 1,
      /* P */ 3, /* Q */ 10, /* R */ 1, /* S */ 1, /* T */ 1,
      /* U */ 1, /* V */ 4, /* W */ 4, /* X */ 8, /* Y */ 4,
      /* Z */ 10};

public static int letterScore2(char c) {
    char cAsUppercase = Character.toUpperCase(c);
    int index = cAsUppercase - 'A';
    if (index < 0 || index >= 26) {
        // handle error
    }
    return scoresByChar[index];
}
To Dos for Thursday

- Read Ch. 14 (Organizing Straight-Line Code)
- Read Ch. 15 (Using Conditionals)
- Read Ch. 16 (Controlling Loops)
- Read Ch. 17.1 and 17.2 (Multiple returns & Recursion)
  - 17.3+ is optional given that it doesn’t really relate to Java

- Get a jump start for next code review: (not due, but …)
  - Parse a JSON description of a series of rooms
  - Verify that if you can get from A to B, that you can get from B to A.
**JSON format**

- **Direction:**
  - direction: String
  - room: String  // a Room’s name

- **Room:**
  - name: String
  - description: String
  - directions: Direction []

- **Layout:**
  - initialRoom: String  // a Room’s name
  - rooms: Room []
```json
{
    "initialRoom": "MatthewsStreet",
    "rooms": [
        {
            "name": "MatthewsStreet",
            "description": "You are on Matthews, outside the Siebel Center",
            "directions": [
                {
                    "direction": "East",
                    "room": "SiebelEntry"
                }
            ]
        },
        {
            "name": "SiebelEntry",
            "description": "You are in the west entry of Siebel Center. You can see the elevator, the ACM office, and hallways to the north and east.",
            "directions": [
                {
                    "direction": "West",
                    "room": "MatthewsStreet"
                },
                {
                    "direction": "Northeast",
                    "room": "AcmOffice"
                },
                {
                    "direction": "North",
                    "room": "SiebelNorthHallway"
                }
            ]
        }
    ]
}
```