Examiners' commentary 2017–2018

CO1109 Introduction to Java and object-oriented programming – Zone B

General remarks

The examination was attempted well by the majority of candidates, with some candidates showing an excellent grasp of Java that went beyond the basics. It was good to see that many candidates, in writing classes, divided the work among methods. These candidates normally achieved first class marks. A minority of candidates would have been well advised to do some basic preparation for the exam including reading the subject guide (both volumes) and attempting the exercises within it. Programming cannot be learned without practice.

Comments on specific questions

Question 1

a. (i) The most appropriate kind of loop should have been as follows:

(A) `while`

(B) `for`

(C) no loop needed

(D) nested `for` loops

Some candidates need to note that if/else statements are not loops. In addition many candidates gave their answer to (A) as 'no loop needed', perhaps assuming that the input would always be correct on the first attempt, which is unlikely. A while loop is a better answer, as this allows invalid input to be rejected, and valid input to be solicited until received. Many candidates gave in answer to (C) that a loop of one type or another was needed, but the first entry in an `Array`, *a,* can be directly accessed, *e.g.* `System.out.println(a[0])` – no loop is needed for this statement to work. Only a minority answered that the best solution to (D) was nested loops.

(ii) The majority of candidates chose the correct answer from the given multiple-choice options. The correct answer for part (ii), was:

(A) The initialisation expression

(iii) The correct answer for part (iii), was:

(A) Infinite loop – the program will hang with no output.

A large number of candidates answered this incorrectly, with many thinking that the class would not compile. Candidates should refer to the subject guide, section 8.4, which explains that `for(;;)` is syntactically correct.

b. The correct number of asterisks (*) to be printed should have been as follows:

(i) 10 asterisks

(ii) 8 asterisks

(iii) 8 asterisks

(iv) no output

(v) Infinite loop

(vi) 6 asterisks

(vii) 3 asterisks

(viii) Infinite loop

(ix) Infinite loop

The majority of candidates answered all sub-questions (i-ix) for part (b) correctly. Common mistakes were off by one error and thinking that an Infinite loop had no output, or that (iv) a loop with no output, was Infinite loop.

c. The missing two methods in the *Verifier* class should have been as follows:

```
private static String getUserInput(String msg){
   System.out.print(msg);
   return scanner.nextLine();
}

private static boolean isValidCustomer(String customer)
{
   for (int i = 0; i < customers.length; i++) {
      if (customer.equals(customers[i])) {
         return true;
      }
   }
   return false;
}
```

Almost all candidates understood that they needed to write the *getUserInput()* and *isValidCustomer()* methods. Some made mistakes by being too specific, and writing code that would not work in general. This included the guard `i < 4` in the *isValidCustomer()* `for` loop. This is fine until items are added or removed from the `Array`, changing the length. In addition some candidates had their *isValidCustomer()* method compare the search `String` to each item in the `Array` by name, *e.g.* four statements such as:

```
if (customer == "Bob") return true;
//3 more statements for "Rob", "Jing" and "Ahmed".
```

There were a few mistakes with the method types, such as making the *isValidCustomer()* method of type `String` when it needs to return a Boolean value, or writing `void` methods that returned nothing. Another common error was having the *getUserInput()* method print a specific message (usually along the lines of "Enter customer name"), when it should take a `String` parameter and print the parameter as its message to the user, or firstly reading in user input and secondly asking for user input. However, by far the most common error was with the logic of the `for` loop in the *isValidCustomer()* method, where candidates wrote loops that would return true or false based either on the first item in the `Array`, or the last, as follows:

- With each iteration of the loop, if the search item is found return `true` else return `false`. This means that the first iteration of the loop ends the method, since it will always return either `true` or `false`, and since once a method returns something, the method ends. Hence the search item can only be found if it is the first item in the `Array`.

- With each iteration of the loop a `boolean` variable is made `true` if the search item is found, and `false` if it is not found. Each iteration of the loop overwrites the value in the `boolean` variable set by the previous iteration. The final iteration of the loop will determine whether the `boolean` is true or false when the loop ends. Hence the search item can only be found if it is in the last place in the `Array`.

Question 2

a. In part (a) it was not unusual for candidates to answer each part incorrectly.

(i) The correct answer from the given multiple-choice options was:

(B) Correct the first error only and recompile

Less than five percent of candidates answered part (i) correctly. Most thought that the right answer was to correct all errors and recompile. The subject guide, volume 1, section 2.8.1, states:

> *The best way to correct [compilation errors] is just to correct the first one and then recompile. This is because the first error sometimes makes the compiler think there are lots of other errors which are not really there.*

(ii) This part was answered correctly by most candidates. The correct answer from the given multiple-choice options was:

(B) *Class* should be *class*

(iii) The correct answer from the given multiple-choice options was:

(A) `'else' without 'if'`

Many candidates chose (C), `unreachable statement`, or (D) *None of the above* as their answer. Most did not understand that without brackets enclosing the statements: `x=1;` and `y=2;` the `if` statement ended with `x=1;` Therefore, as far as the compiler was concerned, the `else` statement was separated from the `if` statement by the `y=2;` statement, giving rise to the compilation error:

```
'else' without 'if'
```

(iv) This part was answered correctly by most candidates. The correct answer from the given multiple-choice options was:

(C) `return type required`

b. The correct 'yes' or 'no' answers to statements (i-ix) are as follows:

(i) NO

(ii) NO

(iii) YES

(iv) YES

(v) YES

(vi) YES

(vii) YES

(viii) YES

(ix) YES

Many candidates answered part (b) entirely correctly, with all candidates achieving most of the marks for the question. There were no common errors.

c. Various correct answers were possible; including the following model answers:

```
public static String sentenceCase(String s) {
   String p = s.toUpperCase();
   String q = p.charAt(0) +
      p.substring(1).toLowerCase();
   return q;
}
   //substring(1) gives all chars from the second char
   to the end of the String, ie all chars except the
   first.
```

or

```
public static String sentenceCaseX(String s){
   String a = "";
   return s.substring(0,1).toUpperCase() +
      s.substring(1).toLowerCase();
   }
//substring(0,1) gives the first char of the String
```

Common errors included giving the wrong parameters to the `String.substring()` methods. For example `substring(0,0)` returns the empty `String`, not the first `char` of the `String` as candidates probably intended. Another common error with `String.substring()` parameters was to use `substring(1, s.length()-1)`, to get the `String` *s* without its first `char`, but this will return the `String` without its first and last `char`; note that `substring(1, s.length())` would be a correct way to get `String` *s* without its first `char`, although `s.substring(1)` would be a better since it would be harder for an error to creep in.

Syntax errors with `String.charAt(0)`

Candidates often wrote logically correct methods with syntax errors. For example, many candidates made the error of attempting to take the first letter of the word using `String.charAt(0)` as follows:

```
String t = word.charAt(0);
```

Which gives a compilation error as the `String.charAt()` method returns a `char`, and a `char` value cannot be assigned to a `String` variable.

Another error was:

```
String t = (s.charAt(0)).toUpperCase();
```

There are two errors above, firstly an attempt to assign a `char` to a `String`, and second attempting to use a `String` method, *toUpperCase(),* on a `char`.

A minority successfully used `charAt(0)` by implementing `String` concatenation, *e.g.*

```
String t = "" + word.charAt(0);
```

or

```
String t = ""; t += word.charAt(0);
```

This works as a `String` can be concatenated with any primitive variable or value using the + operator.

This was a hard question, and those candidates who wrote correct answers under exam conditions are congratulated.

Question 3

a. (i) The programme will not compile because `3/1.2 < 0` should be contained in brackets, *i.e.* `(3/1.2 < 0)`. Most candidates answered this correctly, although a minority thought that the problem was that there would be a compilation error (1) because 0 is not greater than 3/1.2, or (2) because of comparing an `int` value to a `double`.

(ii) Candidates answered this part correctly on the whole, however a minority thought that the answer was that the class would not compile. The correct answer from the given multiple-choice options was:

(B) `goodbye`

(iii) This part was answered correctly by most candidates. The correct answer from the given multiple-choice options was:

(A) `true`

(iv) This part was answered correctly by most candidates. The correct answer from the given multiple-choice options was:

(B) The program will output nothing at all.

b. The correct answers to parts (i-iii) are as follows:

(i) 3+3

6

33

(ii) 2

2.5

2.5

(iii) /*1*/ and /*2*/

This question was answered correctly by the majority, with one common error, a minority thought that that answer to (ii) was:

```
2
2.5
2.0
```

c. A model answer for this would be as follows:

```
import java.util.Scanner;

public class TFQuestions{
   public static String s = "Enter 'true' or 'false' to answer:";

      public static void main(String[] args){
         question();
   }
   public static void question() {
      Scanner scanner = new Scanner(System.in);
      String input = "";
      System.out.println("Human beings glow in the dark.");
      System.out.println(s);
      input = scanner.nextLine();
      boolean answer1 = Boolean.parseBoolean(input);
```

```
        System.out.println("A piece of paper can only be folded
           in half 8 times.");
        System.out.println(s);
        input = scanner.nextLine();
        boolean answer2 = Boolean.parseBoolean(input);

        if(answer1 && answer2) System.out.println("You are so
           right");
        else if (!answer1 && !answer2)System.out.println("Wrong
           answers!");
        else System.out.println("You are half right");
    }
}
```

This question was intended to test candidates' grasp of Boolean logic. Very few candidates gained full marks for this question with many varied individual errors seen, including syntax errors such as trying to assign the value returned by the `Boolean.parseBoolean()` method to a `String`, or forgetting to declare `boolean` variables. Examiners tended to be forgiving of syntactical errors if the logic was correct, which it often was, although some candidates had a tendency to overcomplicate their answers.

A number of candidates used an `int` value to determine if users' answers were true or false, such that the `int` might be 0 if both answers were false, 1 if one was true and one false, and 2 if both were true. The output could then be given correctly and simply with 3 `if` statements, but many candidates used complicated if/else/if statements, unnecessary as there is no overlap between 0, 1 and 2.

Question 4

a. Almost all candidates answered all parts of this question correctly. The correct answers from the available multiple-choice options to parts (i-iv) are as follows:

(i) (B) When the *bling()* method reads in the end of file character

(ii) (D) The program will output the text contained in the text file 'filey.java'

(iii) (B) Nothing would appear on the screen as the condition `t!=-1` would be false at the at the beginning, since the end of the file would be found by the *filey* program immediately.

(iv) (C) `FileNotFoundException`

b. The correct answers to parts (i-iv) are as follows:

(i) `division done`

(ii) `zeta`

(iii) (E) None of the above

(iv) `NumberFormatException`

Parts (i), (ii) and (iii) were answered correctly by most candidates. A few candidates struggled with the answer to (iv) writing such things as 'InputMismatchException', which at least showed that these candidates had the right idea, even if their examination preparation could have been better.

c. A model answer for this would be as follows:

```
public static void multiplyFileContents() {
   int x;
   try {
      Scanner in = new Scanner(new
         FileReader("file1.txt"));
      PrintStream out =new PrintStream(new
         FileOutputStream("file2.txt"));
   while(in.hasNextLine()) {
      String line = in.nextLine();
      x = Integer.parseInt(line);
      x = x * 10;
      out.print(x+"\n");
   }
   in.close();
   out.close();
   } catch (Exception e){
      System.err.println("Error: Could not open or
         save");
   }
}
```

In this question candidates needed to read in each line of the *file1.txt* file as a `String`, parse each `String` read in to an `int`, multiply the `int` by 10, then save the result to another text file. Only a very small minority could correctly write statements to open, read and parse `Strings` from a text file, to open and write `Strings` to a text file, and to close the files after the reading in/writing to loop.

Many candidates lost marks by copying the statements in the *bling()* method that read in `chars`. Candidates can find information about reading in from a file line by line in section 7.6.3 of the second volume of the guide. Many mistakes were also seen in streams to write to the file, candidates can consult section 7.6 of volume 2 of the subject guide for information about writing to a file, and why we should always close the file after writing to it (to be sure that our data has been saved).

Mistakes were also seen in using try/catch, principally not including all statements in the try block that could cause errors. For example, only including the statements to open the file in the try block.

Question 5

a. It was rare for any candidate to achieve full marks for part (a).

(i) There is a simple rule to distinguish primitive from reference variables in Java – all reference variables start with an upper case letter. Applying this rule gave the answer to (i) as `Boolean` and `String`. Many candidates gave `String` as their only answer, since most candidates seemed to think that `Boolean` was a primitive variable, with one or two candidates even writing comments to the effect that `Boolean` was a primitive variable and the capital letter it began with was an error on the paper (it was not).

(ii) The answers for part (ii) are as follows:

(A) FALSE

(B) TRUE

Again only a minority gave the correct answer, with most candidates thinking that (A) was true.

(iii) The output of the *Int*Param class will be:

```
3
```

This was usually answered correctly, showing that candidates understood that giving an int variable as a method parameter did not change the value in the int variable.

(iv) The output of the *Array*Param class will be:

```
5
```

This part was also usually answered correctly, demonstrating that candidates understood that the `Array` *is* changed by the *p()* method. This is because the method receives a copy of the pointer to the array, and it uses this copy to find and change the values pointed to. Hence from the point of view of the main method, the `Array` reference variable is the same, but the place pointed to in memory has had its values accessed and changed by the *p()* method.

b. (i) The correct answer to part (i) was:

```
(char)c
```

Many candidates gave answers to (i) that focused on exception handling, and gained no credit. In the statement System.out.println("You typed in "+(char)c); an int variable c, was being cast to a char with (char)c.

(ii) The correct answer from the given multiple-choice options was:

(A) If the user enters 'k' the output will be '107'

This was answered correctly by a minority of candidates, with many choosing (C) if the user enters 'k' the output will be 'k' or (D) None of the above as the correct answer.

(iii) The model answer for part (iii) is as follows:

```
public class Unicode{
   public static void main(String [ ] args){
      System.out.println((char)99);
   }
   }
```

This was normally answered correctly, although some candidates struggled with the syntax, even though it could have been partly copied from part (i). Note that the answer to this question can also be found in section 6.6 of volume 2 of the subject guide.

(iv) The correct answer from the given multiple-choice options was:

(A)
```
120
X
```

Here, candidates were asked for the 'most likely output of the Chars class', and usually answered (B) [6B on the first line and x on the second] or (C) No output – compilation error. (A) is the correct answer because it is the only answer with an int followed by a char, and hence is the most likely output of the Chars class. The reason why (A) is the most likely output can be found in volume 2 of the subject guide, section 6.6.

c. A model answer for this would be as follows:

```
import java.util.Scanner;

public class Q5Answer {

   public static void askUserForNumber(String input) {
      int number = 0;
      boolean validInput = false;
      Scanner scanner = new Scanner(System.in);
      while (!validInput) {
         try {
               number = Integer.parseInt(input);
               validInput = true;
         }
         catch ( Exception e) {
               System.out.print("\""+input+"\" is not a valid
                        number. Enter a number: ");
               input = scanner.nextLine();
         }
      }
      if (number < 100) System.out.println
            ("Your number is small.");
      else if (number > 1000000) System.out.println
            ("Your number is big.");
      else System.out.println
            ("Your number is nothing special.");
   }

   public static void main(String[] args) {
      askUserForNumber(args[0]);
   }
}
```

The majority of students lost marks on this question by ignoring the command line input (`args[0]`), and starting by asking the user for a number. This was heavily penalised by the examiners, as it seemed to be avoiding the most difficult part of the question. It was very challenging to implement testing the command line input to see if it could be parsed to an `int`, accepting it if it could, rejecting it if not and entering a loop asking the user for correct input.

Some correct answers were seen, most of which varied in small ways since various correct answers were possible.

Question 6

a. The correct 'true' or 'false' answers to statements (A-H) are as follows:

(A) Every constructor must have the same name as the name of the class. **TRUE**

(B) Constructors do not have return types. **TRUE**

(C) A constructor is typically used to give values to the object's instance variables. **TRUE**

(D) A class can have up to three constructors. **FALSE**

(E) When one class extends another, the keyword super can be used in a constructor of the extending class to access a constructor in the class that is being extended. **TRUE**

(F) An instance method has the keyword static in its heading. **FALSE**

(G) Person extends SentientBeing means that the Person class has all the fields and instance methods of the SentientBeing class. **TRUE**

(H) An inheriting class can redefine instance methods from its parent class by overriding them. **TRUE**

Candidates gave answers to part (a) that were mostly or entirely correct. No common errors were seen.

b. Most candidates gave a correct answer, either giving the fragment numbers in the correct order, or writing their answers out in full. It was pleasing to see that most candidates correctly answered this question.

For full credit, the fragment numbers should have been in one of the following orders:

2, 5, 10, 8, **6**, **7**, **4**, **9**, 3, 1, 11 ***or***

2, 5, 10, 8, **4**, **9**, **6**, **7**, 3, 1, 11

since the order of the instance methods *isDone()* and *toString()* is arbitrary.

Most candidates wrote out their answer in full, as follows:

```
2. public class ToDoList {
      private String item;
      private String importance;
5.    private boolean done;
10.   public ToDoList(String item,String i,boolean done){
8.          this.item = item;
            this.importance=i;
            this.done = done;
      }
6.    public boolean isDone() {
7.          return done;
      }
4.    public String toString(){
            String s = ("Action: "+item+" with ");
            s = s+(importance+" importance");
            s = s+("\nAction taken status: "+done);
9.          return s;
      }
3.    public static void main(String[] args) {
            //test statements
1.          ToDoList toDoList = new ToDoList("Submit 109
                  cwk","high",true);
            System.out.println(toDoList);
      }

11. }
```

c. A model answer for this would be as follows:

```
public class IntExists extends Exists{
   public boolean isContained(int[]x, int y){
      for (int i=0; i<x.length; i++){
         if (y == x[i]) return true;
      }
      return false;
   }
}//other (similar) correct answers are possible
```

Some mistakes were seen with the logic or syntax of the *isContained()* method, but it was good to see that many candidates correctly extended *Exists* and implemented *isContained()* in their new class.