Examiners' commentary 2017–2018

CO2220 Graphical object-oriented and internet programming in Java – Zone A

Comments on specific questions

Question 1

a. The correct statements from the given multiple-choice options were:

   i. (B) Constructor 1 is not valid because the call to this must be the first statement if used in a constructor.

   ii. (A) Constructor 1 is not valid because the call to super must be the first statement if used in a constructor.

   iii. (C) Constructor 3 is valid.

   iv. (C) Constructor 4 is not valid because the variable i has private access in Boo so cannot be directly accessed in SonOfBoo.

Very few candidates answered (i) correctly. Most thought the answer was (A), *this* and *super* cannot be used in the same constructor. In fact, *this* can be used with *super* if it is used for disambiguation, making (B) the best answer. Where *this* is used to call another constructor in the same class it must be the first statement (and cannot be used with *super*). Most candidates answered (ii), (iii) and (iv) correctly.

b. Answers:

   i. An appropriate answer to this would have been because in classes with no constructor, the JVM adds the default no-argument constructor. Most candidates answered this correctly, but those that did not were clearly guessing, writing such things as 'because the class is instantiated in main', or 'because the class has getters'.

   ii. The output of the Fairy class was:

```
null
false
```

   This was usually answered correctly, with candidates demonstrating that they understood that uninitialised instance variables have a default value, although a minority wrote that there would be no output as the variables were not initialised.

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

```
public void setSound(String s){
sound = s;
}

public void setImmortal(boolean i){
immortal = i;
}
```

```
public static void main(String[ ] args){
Fairy fairy = new Fairy();
fairy.setSound("Tinkle");
fairy.setImmortal(true);
System.out.println(fairy.getSound());
System.out.println(fairy.getImmortal());
}
```

This was mostly answered correctly. There was one common mistake that achieved some marks, which was to write a 2-argument constructor and add `Fairy fairy = new Fairy("Tinkle", true);` in the main method. This answer did not achieve full marks because candidates were asked to write methods for the *Fairy* class, not a constructor, and writing a constructor overwrites the default no-argument constructor, meaning that the first line in the main method would then cause a compilation error.

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

```
public class Book extends Novel{
     private String ISBN;
     public Book(String title,String author,String ISBN,
       String publisher){
          super(title, author, publisher);
          this.ISBN = ISBN;
     }
       public String getISBN(){
          return ISBN;
     }
}
```

This was answered well, with the majority of candidates receiving full marks, although some candidates did not give the String instance variable an access modifier or made it *public* (should be *private*), but this was ignored in the marking. All candidates remembered to extend *Novel*, included a String *ISBN* instance variable and wrote a correct *getISBN()* method. Some mistakes were made with the constructor that were penalised in the marking, including: putting the call to the super class constructor as the second statement in the *Book* constructor (a compilation error); writing a 3-argument constructor that did not include the *ISBN* variable; writing a constructor with the *ISBN* variable as the only argument; and writing an empty constructor with no arguments. One student wrote a comment in their (correct) answer that the *ISBN* variable should be lower case, as only final variables should be entirely in upper case, which was quite right.

Question 2

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

(A) TRUE

(B) TRUE

(C) FALSE

(D) TRUE

(E) TRUE

(F) TRUE

Almost all candidates answered this correctly apart from (A), which was considered to be false. Perhaps candidates thought that primitives cannot be added to `ArrayLists`, as `ArrayLists` cannot be parameterised to primitive types. However primitive types can be added to `ArrayLists`, for example this will compile (with a least Java 8):

```
ArrayList<Integer> eg = new ArrayList<>();
eg.add(8);
System.out.println(eg.get(0));
```

ii. The correct 'true' or 'false' answers to statement (A & B) are as follows:

(A) TRUE

(B) FALSE

Usually this was half right, as candidates tended to answer that either both (A) and (B) were true, or both were false.

b. i. Error: *Lex* is abstract; cannot be instantiated / the main method is missing a closing bracket after 'args'.

There was a mistake on the paper, in that the main method in (i) was missing a closing bracket after 'args'. Hence the examiners accepted as the correct answer either the missing bracket, or that *Lex* could not be instantiated. Some candidates wrote *Lex* could not be 'initialised', and this answer was also accepted. A large minority received no marks for answering that the *replace()* method was abstract and needed to be implemented.

ii. The 'true' statement from the given options was:

(A) The *Simon* interface will produce a compilation error because the *toString()* method is not abstract.

Despite being told that only one of the statements in (ii) was true, many candidates stated that both (A) and (B) were true, while some thought all statements were false, and some that all of them were true. A minority correctly answered that only (A) was true.

iii. The correct answer was that *Computer, Laptop* and *SteamTrain* will compile. *Truck* **will not**. ***Or*** 1, 2 and 4 will compile, 3 **will not**. This was quite challenging, but many candidates answered completely correctly, and others achieved at least half marks. Almost all candidates understood that *Truck* would not compile (because it does not implement all abstract methods of *Machine*).

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

```
interface queue{
   abstract queue append (queue q);
   abstract Object first();
   abstract boolean isin(Object o);
   abstract boolean isEmpty();
}//note abstract is implicit so can be missing
```

Where candidates made mistakes with part (c) it was often with the parameters, with the most common error adding a *queue* to every parameter list. Some did not understand that the methods would be instance methods, only invoked with a *queue* object, and acting on that *queue*, and as such did not need a *queue* parameter in general. The exception to this was *append(),* which had to append to *this queue* a new *queue*.

Some candidates gave correct answers except for the *append()* method, which was *void*. In fact *append()* needs to be of type *queue*, as it has to return the new *queue* that it makes – it would not be a very useful method if it did not.

Question 3

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

(A) TRUE

(B) TRUE

(C) FALSE

(D) TRUE

(E) TRUE

(F) TRUE

(G) FALSE (it has 5)

(H) TRUE

This was answered correctly most of the time, with a few mistakes but no candidate giving less than 5 correct answers. In particular (C) and (G) were always correct.

b. i. The correct option for this was (B) The top left corner. This was answered well by all candidates.

ii. This was answered correctly by all candidates. The correct statements here would be:

```
X = X - diameter/2;
Y = Y - diameter/2;
```

iii. A `JButton` instance variable is already in the class, i.e. `JButton dButton;`

Candidates should use the `JButton` instance variable in the following statements, which they should add to the *go()* method:

```
dButton=new JButton("Click me to resize the circle.");
frame.getContentPane().add(BorderLayout.NORTH, dButton);
```

Again, this was answered well by all candidates, where the majority noticed and used the *dButton* instance variable in their answer.

iv. The following statement should be added to the *go()* method:

```
dButton.addActionListener(new DiameterListener());
```

This was sometimes answered with `dButton.addActionListener(this);` which received no marks. The examiners, in general, overlooked minor syntax errors provided that the candidate's intent was clear.

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

```
class DiameterListener implements ActionListener{
      public void actionPerformed (ActionEvent e){
            X = X + diameter/2;
            Y = Y + diameter/2;
            //above two statements restores the
            coordinates to what they were before
            centering. NOTE they are centered in the
            mouseClicked(MouseEvent) method.
            diameter= r.nextInt(drawPanel.getWidth());

            X = X - diameter/2;
            Y = Y - diameter/2;
            //above two statements re-centre the
            coordinates, using the new diameter.
            drawPanel.repaint();
      }
}
```

The answer given above demonstrates one way of answering the question; other correct answers were seen. One thing all answers had in common was making a new random number for the *diameter* variable. Candidates were expected to use `drawPanel.getWidth());` to limit the random *diameter*, and most did, but some used a number, often 600. Candidates were also expected to use the `Random` instance variable, `r,` and most did.

Question 4

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

(A) TRUE

(B) TRUE

(C) FALSE

(D) FALSE

(E) TRUE

It was rare to see completely correct answers here, but there was no pattern to the errors, except that most candidates knew that (C) was correct.

ii. Class *Good* is *final* and cannot be extended by class *Better*.

Perhaps only half of all candidates answered part (ii) correctly, with about a quarter of candidates writing that the *greeting()* method could not be overridden by *Better* because *Good* was a *final* class. Many other different wrong answers were seen that were clearly guesses from unprepared candidates.

b. i. The correct 'true' or 'false' answers to statements (A-C) are as follows:

(A) TRUE

(B) FALSE

(C) TRUE

This was answered correctly by the majority, although a minority thought that (B) was true.

ii. This was answered correctly by the majority of candidates. The output of the class when it is run should be as follows:

```
11-22-44
hello – End of output of first System.out.println()
11
22
44 – End of output of second System.out.println()
```

iii.
```
Integer intObj = new Integer(j); //correct answer
int k = Integer.valueOf(intObj);//unwrapping
i = 5; //autoboxing, converting an int to an Integer
j = i;//unboxing, converting an Integer to an int
```

Correct answers for part (iii) were rare as candidates confused wrapping with unwrapping, autoboxing and unboxing. Wrapper classes, broadly speaking, turn a primitive into a reference variable, or object. Hence the first line, and correct answer, is wrapping the `int` variable *j*, to an `Integer` object, *intObj*. The second line is unwrapping the *intObj* variable into an `int`, or primitive, variable. Boxing and unboxing (introduced in Java 5) is when Java does the conversion between primitives and their object wrapper types automatically. Hence in the third line the primitive 5 is assigned to the `Integer` variable *i*, with Java performing the conversion, and in the fourth and final line, the `int` variable *j* is assigned the value contained in the Integer variable *i*, with Java converting *i* to an `int` in order to make the assignment.

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

```
int id = Integer.parseInt(data.get(0));
String name = data.get(1);
String gender = data.get(2);
String phone = data.get(3);
String email = data.get(4);
int prog = Integer.parseInt(data.get(5));
//int prog = Integer.valueOf(data.get(5))
//alternative way to assign a value to the prog variable
boolean offerMade = Boolean.parseBoolean(data.get(6));


return new ProspectiveStudent(id, name, gender,
      phone, email, prog, offerMade);
```

Some candidates made no attempt at this part of the question, while those who did often lost some marks by confusing `Array` and `ArrayList` syntax (for example writing `int id = Integer.parseInt(data.[0]);`, or by not assigning the `boolean` variable correctly. This was a challenging question, so the minority who achieved full marks are to be congratulated.

Question 5

a. i. Most candidates answered this correctly. The correct statement was:

(C) Because using a buffer is more efficient as reading/writing to file is expensive compared to manipulating data in memory.

ii. The correct answer here was by using the BufferedWriter's *flush()* method. Only a minority of candidates knew the correct answer, with many obvious guesses seen.

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

(A) TRUE

(B) FALSE

(C) TRUE

(D) FALSE

This was usually answered only partly correctly, with very few candidates achieving full marks. Errors seemed to be fairly randomly distributed, suggesting that many candidates were guessing.

b. i. The correct statement was:

(A) An object's state, given by its instance variables is saved. Any objects that are referenced by the instance variables, and in turn any further objects referenced by their instance variables, *etc.* are also saved.

Most candidates knew that (A) was the correct answer, although a minority put (B), suggesting that these candidates did not know that static variables are not saved when an object is serialized. When an object is deserialized static variables are set to their default value.

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

(A) FALSE

(B) TRUE

(C) TRUE

(D) FALSE

This was rarely completely right, with no clear pattern to the errors, suggesting that many candidates were guessing.

iii. An appropriate answer to this would be when attempting to cast the deserialized object to its type (or supertype). To achieve any marks for part (iii) candidates had to say whether the `ClassNotFoundException` would be thrown in the serialization or in the deserialization process. A minority of candidates who were vague on this point received no marks.

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

```
public static ArrayList<String> deserialize(String filename){
  try{
       ArrayList<String> notes = null;
       ObjectInputStream in = new ObjectInputStream(new
              FileInputStream(filename));
       notes = (ArrayList<String>) in.readObject();
       in.close();
       return notes;
  }
  catch (Exception e) {
      System.err.println("Error reading "+filename+". " + e);
      return null;
  }
}
```

Completely correct answers were rare. Candidates used very poor syntax, did not use an `ObjectInputStream` and/or `readObject()`, tried to read from the file as if it were a text file, did not cast, did not return anything, wrote vague error messages in the catch block, did not close streams and forgot to parameterize the `ArrayList`. Less than 20% of candidates attempting question 5 achieved full marks for this part of the question.

Question 6

a. i. Almost all candidates knew that the correct statement was:

(A) Unchecked exceptions are usually due to faulty code logic, rather than unpredictable or unpreventable conditions that arise when the code is running. As such, they should be taken care of by writing classes that are logically correct.

ii. The checked exceptions were:

`ClassNotFoundException`

`EOFException`

`FileNotFoundException`

`IOException`

`MalformedURLException`

`SocketTimeoutException`

Most candidates could not identify all the checked exceptions in the list given. Some could only name one or two. `EOFException` and `SocketTimeoutException` were particularly likely to be wrongly considered as unchecked exceptions by candidates.

b. i. A few candidates gave only one answer to part (i), but the majority were able to identify the two correct statements as:

(A) the thread is waiting for data

(B) the thread is trying to access a locked object

ii. The majority of candidates correctly identified the words that can be used to describe genuine thread states as:

NEW

RUNNING

iii. Again, a few candidates gave only one answer to part (iii), usually just (c). But the majority were able to identify the two correct statements as:

(B) because concurrency is restricted because other threads are forced to wait for their turn

(C) because synchronization brings with it the hazards of thread deadlock (where each thread has the key that the other needs)

c. The three missing code fragments were:

1. `Socket socket = new Socket(host, port);`
2. `BufferedReader reader = new BufferedReader(new InputStreamReader(socket.getInputStream()));`
3. `socket.close();`

Very few completely correct answers were seen, but plenty of obvious guessing by unprepared candidates was, particularly for fragments 1 and 2.