Examiners' commentary 2017–2018

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

Comments on specific questions

Question 1

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

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

   ii. (B) Constructor 2 is not valid because the keyword *this,* when used in a statement to call another constructor in same class, must come first.

   iii. (D) Constructor 3 is valid.

   iv. (A) Constructor 4 is not valid because the variable *s* has private access in *Woo* so cannot be directly accessed in *SonOfWoo*.

   Most candidates thought the answer to (i) was (A), *this* and *super* cannot be used in the same constructor. In the first constructor both key words are being used to call another constructor, which the compiler will not allow. However, it is not true to say that in general *this* and *super* cannot be used in the same constructor, since *this* can be used with *super* if *this* is used for disambiguation, making (B) the best answer.

   Most candidates answered (ii) correctly, but not (iii) where many candidates did not understand that constructor 3 is valid. Candidates perhaps thought that constructor 3 was not valid as it had a float parameter, and there are no instance variables in either *Woo* or *SonOfWoo* of that type. However, the compiler does not care what is in the parameter list, provided that it is syntactically correct, and the single statement in constructor 3, the call to the superclass constructor is valid, since *Woo* has a constructor that takes a single int parameter.

   In part (iv), most candidates did not understand that the statement `s = "woo";` will produce a compilation error since *s* has private access in *Woo*, meaning that it cannot be directly accessed outside the class. Normal practice, if access is needed to instance variables, is to write public getter and setter methods to allow instance variables to be viewed (by getters) and changed (by setters).

b. i. An appropriate answer to this would have been because you can make an object because of the default no-argument constructor. Most candidates answered this correctly, but others wrote such things as 'because sound and age have default values a constructor is not needed'.

   ii. The output of the *Orc* class was:

   ```
   null
   0
   ```

   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. Some candidates were confused about default values, thinking the default value for an `int` is `null`, and for a `String` is the empty `String`. In Java reference variables (objects)

have the default null, while primitives have a literal value. For example, booleans have the default value false, while Booleans default to null. String is a reference variable, so its default value will be null, while int is a primitive, so its default value is 0.

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

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

public void setAge(int howOld){
      age = howOld;
}

public static void main(String[ ] args){
      Orc orc = new Orc();
      orc.setSound("Grunt");
      orc.setAge(102);
      System.out.println(orc.getSound());
      System.out.println(orc.getAge());
}
```

Most candidates wrote correct setters for the *Orc* class, but not all candidates used their getters to set the value of the *sound* and *age* variables, instead assigning their values directly in the main method, for example with age = 102;. Since the question was worded badly enough to allow this, and since this would work as the private variables were being accessed inside their own class, these candidates received full marks.
A minority of candidates wrote a 2-argument constructor and added `Orc orc = new Orc("Grunt", 102);` (or similar) in the main method. This answer did not achieve full marks as (1) candidates were asked to write methods for the *Orc* class, not a constructor, and (2) 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 AudioBook extends Novel{
      private String narrator;

      public AudioBook(String title, String author,
      String narrator, String publisher) {
      super(title, author, publisher);
      this.narrator = narrator;
   }

   public String getNarrator(){
         return narrator;
}
}
```

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. The most common mistakes were writing a one-argument constructor that just set the value of the Narrator variable, and writing a 3-argument constructor, that did not include the Narrator variable. Almost all candidates wrote a correct getter for the `String` variable, which was in contrast to part (b)(iii) where many mistakes were made with the setters, including wrong return type (should be void), no parameters and returning a value or variable. These mistakes implied that a minority of candidates did not understand the difference between getters and setters. A setter, such as *setSound()* in part (b)(iii), gives a new value to an instance variable, so it must take that variable as a parameter, and should be of type `void` since it does not need to return anything. A getter needs no parameter as nothing is being changed, it must have the same type as the instance variable whose value it is returning, and, of course, it must return that variable.

Question 2

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

(A) TRUE

(B) TRUE

(C) TRUE

(D) FALSE

(E) FALSE

(F) FALSE

Almost all candidates correctly thought that (A) and (B) were true, but wrongly considered that (C) was false. Some candidates wrote comments about (A) pointing out that the `ArrayList.get()` method returns an object, that is only unwrapped if assigned to a primitive, which is correct. The automatic wrapping and unwrapping is known as boxing and unboxing, which is not a feature that belongs to `ArrayLists`, but is something implemented since Java 5 to automate wrapping and unwrapping throughout Java.

Most candidates understood that (D) was false but there was some confusion about (E) and (F). (E) is false because while `ArrayLists` can be parameterised, this can only be to reference variables (objects), hence any object that is not of the parameterised type will be rejected by the compiler. (F) is false because `ArrayLists` cannot be parameterised to primitive variables. Perhaps some of the confusion arose because while `ArrayLists` cannot be parameterised to primitives, it is possible to add a primitive to an `ArrayList`, and it will be automatically wrapped (boxed) to the object parameter, if possible. For example, the following, in an otherwise error-free class, will compile:

```
ArrayList<Integer> eg = new ArrayList<>();

eg.add(8);

//eg.add(8.0);
```

Removing the comment marks (//) from the third statement and compiling again will give the following error:

```
error: incompatible types: double cannot be
converted to Integer
```

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

(A) FALSE

(B) FALSE

Usually this was partly or completely incorrect, with candidates answering that both statements were true, or that one was true and the other false. Some candidates correctly answered that (A) was false, but for the wrong reason, some candidates wrote that it was always necessary to check both sides of a `boolean` expression. In fact the short circuit operators are `&&` and `||`, and their use is often appropriate.

b. i. *Q2a* is concrete and should override abstract method *sum()* in *Trev*. This was normally answered correctly, although a few candidates wrote that the error was that an abstract class could not be instantiated, when Q2a is actually implementing an interface, not trying to instantiate an abstract class.

ii. Error in question, all marks given.

iii. Error in question, all marks given.

In part (ii) there was a mistake in the question, in that candidates were asked to say which of the statements were true about the interface *Rex*, when *Rex* was an abstract class. For the potential confusion that this might have caused, all candidates were given full marks for this part of the question, whatever answer they gave, or even if they gave no answer at all. If the question had correctly asked which statements were true about the *Rex* abstract class, the answer would have been that (C), the *Rex* class will compile, is true, and the other statements false.

Part (iii) again had a mistake in the question, in that the *Bicycle, Truck and SpaceShip* classes were implementing the *Vehicle* class, when they should have been extending it, for example:

```
abstract class Bicyle implements Vehicle
//'implements' should be 'extends'
```

This changed the correct answers, making them harder than intended, hence again all candidates were given full marks for this question, whatever their answer, or even if they did not answer (since the examiner could not know the reason for not answering and how it related to the mistake in the question). If *Bicycle* had been extending *Vehicle*, then the correct answer would have been that *Bicycle*, *MountainBike* and *SpaceShip* would compile, but *Truck* would not.

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

```
interface set{
     abstract set intersect (set s);
     abstract set union (set s);
     abstract boolean isin(Object o);
     abstract boolean isEmpty();
}
//note abstract implicit so can be missing
```

Where candidates made mistakes with part (c) it was often with the parameters. The most common errors were adding a *set* to every parameter list, or writing all methods with no parameters at all. Some clearly did not understand that the methods would be instance methods, only invoked with a *set* object, and acting on that *set*. For example, assuming that *alice, bob* and *cariad* are sets, then

```
boolean b = cariad.isEmpty(); would be a valid
statement, as would alice = cariad.union(bob);
```

Question 3

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

(A) TRUE

(B) TRUE

(C) FALSE

(D) FALSE

(E) FALSE

(F) FALSE

(G) FALSE

(H) TRUE

This was answered correctly most of the time, with the only common error being that (E) was often given as true.

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 often answered with very poor syntax, which would not work even if corrected and 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 ways of re-centering the *X* and *Y* coordinates are possible, but none were seen. Candidates were expected to use the `Random` instance variable, *r,* in their answer, but very few did, and many showed great confusion about how to make a random number in Java.

Question 4

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

(A) FALSE

(B) TRUE

(C) TRUE

(D) TRUE

(E) FALSE

Very few candidates knew that (E) is false, and those that did tended to write a comment that making an instance variable *final* is pointless, which is true.

ii. Most candidates gave the correct answer to part (ii) as:

AskToDance() in *Happier* overrides *askToDance()* in *Happy*. However *askToDance()* is a final method and cannot be overridden.

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

(A) FALSE

(B) TRUE

(C) TRUE

This was answered correctly by the majority of candidates.

ii. The output of the class when it is run should be as follows:

```
hello/greetings/wassup
salutation – End of output of first System.out.println()
hello
greetings
wassup – End of output of second System.out.println()
```

Again, this was answered correctly by the majority, although a few got the output of the second `System.out.println()` statement wrong, writing such things as `wassup=salutation`, or giving three lines of correct output, but then incorrectly adding `salutation` to the end. It was hard to see where this confusion was coming from, since the output came from printing the contents of the *r2* `Array`, one to a line, and the *r2* `Array` was made from the first entry in the *r1* `Array`, which these candidates must have known contained `hello/greetings/wassup`, and definitely did not contain 'salutation'.

iii.
```
i = 5; //autoboxing, converting an int to an Integer

//incorrect answers below
j = i;//unboxing, converting an Integer to an int
Integer intObj = new Integer(j); //wrapping
int k = Integer.valueOf(intObj);//unwrapping
```

Some correct answers were seen, but candidates confused autoboxing with unboxing, and sometimes with wrapping and unwrapping. Wrapper classes, broadly speaking, turn a primitive into a reference 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 first line of the answer above the int literal value 5 is assigned to the Integer variable *i*, with Java performing the conversion, and in the second 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);
```

Many candidates lost some marks by confusing `Array` and `ArrayList` syntax (for example writing `String name = data.get[1];`. Others introduced 'off by one' errors, by assigning items out of order, for example by assigning a value to the *id* variable by parsing to an `int` the fifth item in the `ArrayList (data.get(4))`, rather than the first. Oddly, some candidates gave answers that were completely correct, except that they had forgotten that a `String` value could not be directly assigned to an `int`, and had just written `int id = data.get(0);`

Question 5

a. i. Most candidates answered this correctly, although a few candidates thought the answer was (D), none of the above. The correct statement was:

(A) 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 knew the correct answer, with many obvious guesses seen.

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

(A) FALSE

(B) TRUE

(C) FALSE

(D) TRUE

This was usually answered only partly correctly, with most candidates thinking that (A) was true and (B) false. It is not true to say, as (A) does, that the *SourceViewer* class will ignore images, since images are not in the HTML code, but information about images is (i.e. some images may be constructed from HTML code, and others have tags that the class will copy). Certainly the class cannot download image files, but that is not the same as ignoring images.

b. i. Most candidates knew the correct answer, although a few thought that (C), The object's source code, was correct. 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.

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

(A) TRUE

(B) FALSE

(C) TRUE

(D) TRUE

Candidates often thought that (A) was false and that (B) was true.

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. Most candidates lost marks by answering that 'it will happen during serializing and deserializing' or equivalent.

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;
  }
}
```

Very few candidates achieved full marks for this part of the question. Many received no marks at all, because they did not include an `ObjectInputStream` in their answer, and without this there can be no deserializing. Many candidates tried to read from the file in a loop adding items one at a time to the `ArrayList`, none that were correct were seen. In particular candidates tried to use methods such as *hasNext()* or *hasNextLine()* in their loops, but neither the `ObjectInputStream` class nor the `FileInputStream` class have such methods (or any equivalent). These candidates were confusing reading from a text file with deserializing. In addition, the question asked that a single `ArrayList<String>` be deserialized, hence a loop was not necessary to read and cast a single object to an `ArrayList<String>`.

Many candidates only included the attempt to open the file in the catch block, and not the attempt to read and deserialize the object. These candidates must have been unaware that the `ObjectInputStream.readObject()` method could throw exceptions, including `ClassNotFoundException`. The other exceptions the method can throw are `InvalidClassException, StreamCorruptedException`, `OptionalDataException` and `IOException`, but it is only the `ClassNotFoundException` that candidates are expected to be aware of.

Question 6

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

(C) Exceptions should only be used for problems caused by an application's interactions with the outside world, and not by the internal logic of the code. Therefore, exceptions that may arise due to poor logic on the part of the developer are not checked.

ii. The checked exceptions were:

`ClassNotFoundException`

`EOFException`

`FileNotFoundException`

`IOException`

`SocketException`

`UnknownHostException`

Most candidates could not identify all the checked exceptions in the list given, so very few achieved full marks.

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.

(C) 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

RUNNABLE

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:

(A) because method access is slowed.

(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 = serverSocket.accept();`
2. `PrintWriter writer = new PrintWriter(socket.getOutputStream());`
3. `writer.flush(); // or writer.close();`

Most candidates received some marks for part (c) although few completely correct answers were seen. The most common error was failing to open an output stream on the *socket*, by, for example, giving the `PrintWriter` the parameter `socket`, instead of `socket.getOutputStream());`