Examiners' commentaries 2016–17

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

General remarks

The majority of candidates did very well on this examination. However, a significant minority would be advised to ensure much better familiarity with basic programming concepts that can only be achieved by attempting to write and compile statements, methods and classes. Therefore it is important to practise, practise, practise.

Comments on specific questions

Question 1

This was a popular question, chosen by 99 per cent of candidates, with many achieving full credit.

Part a

In (i) 78 per cent of candidates knew that the class would compile and output `true`; 14 per cent thought that the class would not compile; and 8 per cent thought that the class would compile and output 3.

For part (ii) a majority (65 per cent) understood that the *Bool4* class would output nothing. A minority (20 per cent) thought that the program would output `hello`; while a smaller minority – 12 per cent – thought that the class would not compile, some giving the reason that the variable *t* could not be both 0 and 3. The remaining candidates chose 'None of the above' as their answer.

For part (iii) most candidates (94 per cent) wrote a correct answer. Various correct answers were seen, such as a class with an empty main, or a main method with the statement `System.out.println("")` – thus literally outputting nothing. The most common mistake was having a `System.out.println()` statement that did output something; for example, asking the user for input ('enter a number'); which is, of course, output. The other mistake seen was writing a syntactically correct class with no main method; such a class would compile, but it would not run.

Part b

Many Part (b) answers were completely right; others showed one error and (quite often) were mistaken in thinking that a finite loop was infinite. Correct answers and common mistakes (in brackets) follow:

i. 9 (8, 10, infinite) [90 per cent answered correctly]

ii. 3 (infinite, 2) [90 per cent answered correctly]

iii. 5 (4, 3, 6, 0, infinite) [86 per cent answered correctly]

iv. no output (9, infinite, 10, 8, 14, 16, 19) [63 per cent answered correctly]

v. infinite (1, 2, 0) [69 per cent answered correctly]

vi. 20 (19, 21) [86 per cent answered correctly]

vii. 4 (3, 5, 7) [69 per cent answered correctly]

viii. infinite (5) [94 per cent answered correctly]

ix. no output (infinite, 2, 6, 1) [80 per cent answered correctly]

More than one candidate noted that infinite loops are not infinite, but will continue for some finite time, ending when the program crashes due to integer overflow – well, yes, but do you want to have to write that out in full each time you think a loop will continue for a finite amount of time, eventually ending when the program ends with an error?

Part c

The output requested should have been as follows:

```
****
***
**
*
#
#
#
*
**
***
****
```

The majority of answers were completely right. The most common error was putting the #s in a horizontal line, with the rest of the output given correctly. A few thought that the stars were in 4 by 4 squares top and bottom, and some got the triangle shapes given by the stars the wrong way round; namely:

```
   *
  **
 ***
****
```

A very few candidates gave output that was completely wrong; these candidates were clearly guessing and had not prepared very well – if at all – for the examination.

Question 2

Although not quite as popular as Question 1, this question was attempted by about 95 per cent of candidates, and was done well on the whole.

Part a

Part (i) was answered correctly by all, with candidates writing something along the lines of: 'the program was attempting to assign a real number to an `int` variable, causing a compilation error.'

For part (ii) various incorrect answers were given, such as the variable *k* was out of scope in the main method, or *k* should have an access modifier, or *k* should be declared in the main method (the most popular incorrect answer), or that there were in fact no errors. The answer the examiners were looking for (which was given by a minority of candidates) was that since *k* was not a static variable, it could not be accessed in a static context. Full credit was given to equivalent answers, such as since *k* was

an instance variable it could not be directly accessed by main, or that *k* could only be accessed by main through making and using an object of the class.

In part (iii) 98 per cent of candidates understood the error, and answered either that there should be double quotes around 'hello', or, equivalently, that *hello* was an undeclared variable. Each of these responses gained full credit.

Part (iv) was answered incorrectly by 40 per cent of candidates. A good answer would have said that the *add(int, int)* method had no return type. An even better answer would have said that the *add(int, int)* method should have had the `int` return type, as it was returning an `int`. Various wrong answers were seen, with the most popular being that the method should have an access modifier (public and private were variously suggested). Other answers included: the return values should be in brackets; the return statement should not be adding variables; there was no error; a static method cannot be referenced in a non-static context; *x* and *y* are not declared in the method; variable *k* should be declared in the class as `static int k` since it is referenced a `static` method; *x* and *y* cannot be used in the main method as they are out of scope. These answers showed two things: (1) many candidates were guessing because they had no idea what the problem was; and (2) these candidates have a troubling lack of understanding of some fundamental programming concepts. What this suggests is a lack of programming practice. Programming cannot be learned from a book, as concepts only make sense once **applied**. Making mistakes when writing programs can be very frustrating, as errors can be hard to see, but these errors are of great value, as we do not forget the lessons we learn from them.

Part b

In part (b) the correct answers were:

i. NO [92 per cent answered correctly]

ii. YES [98 per cent answered correctly]

iii. YES [93 per cent answered correctly]

iv. YES [92 per cent answered correctly]

v. YES [85 per cent answered correctly]

vi. YES [37 per cent answered correctly]

vii. YES [61 per cent answered correctly]

viii. YES [57 per cent answered correctly]

ix. NO [59 per cent answered correctly]

In part (vi) what candidates failed to appreciate was that giving an index that was too large to the *charAt(int)* method would be a run-time error; since the compiler would find an `int` where it expected an `int`, this would not be a compilation error. Java has an `IndexOutOfBoundsException` class because these sort of mistakes can and do happen.

Part c

In part (c) a quarter of candidates gave answers that were not just wrong, they showed no evidence that the candidate had prepared for the examination. For example, many of these candidates wrote methods with the heading *min(int, int)*; that is, they copied the heading directly from the question paper and did not understand that the parameters needed to be given names. These candidates proceeded to write methods with various statements that would not compile such as `if x < min();`

Various correct answers were possible; including the following:

i.

```
static int min (int x, int y){
   if (x<y) return x;
   else return y;
}
```

ii.

```
static int minOf3 (int x, int y, int z){
   int a = min(x,y);
   int b = min(a,z);
   return b;
}
```

The answer to part (ii) given above reflects the logic of the most commonly seen answer, which was to find the minimum of the first two parameters using the *min(int,int)* method and store the answer in a variable. After this, return the new variable or the third parameter, whichever is smaller, using the *min(int,int)* method again to make the determination. Very few candidates understood that this process could be abbreviated as follows:

```
static int minOf3 (int x, int y, int z){
   return min(x, min(y,z));
}
```

Some candidates lost credit by writing logically correct answers but not giving their methods a return type.

Some candidates got no credit for part (i) because in their answer they did not cover the case of *x* being equal to *y*. The logic of their methods was as follows:

```
static int min (int x, int y){
   if (x<y) return x;
   else if (y<x) return y;
}
```

In the above method what happens when *x* equals *y* is undefined, and the compiler will flag this as a missing return statement.

Part (ii) was marked as if the answer to (i) was correct, so candidates did not lose marks a second time if their answer to part (i) was not completely correct. However, a minority of candidates did lose marks for not using the method written in part (i) in answering this part. Even if the answer to part (ii) was logically and syntactically correct, candidates received no credit if they did not use the answer to (i) as instructed.

Question 3

This question was attempted by 50 per cent of candidates.

Part a

In part (a), the examiners were looking for an answer similar to the following:

```
public static String magic8Ball(){
   Random r = new Random();
   int index = r.nextInt(answers.length);
   return answers[index];
}
```

The examiners expected candidates to understand from the main method given with the question that *magic8ball()* must be a static method, and that it should be of type `String`. Candidates were also expected to understand that since the *answers* array is a class variable it need not be a parameter of the method.

The following mistakes were seen in otherwise correct methods (candidates received all or some of the credit):

- using `answers.length-1` to make the random number. This would give a number between 0 and 5 inclusive, meaning that the final array element could never be chosen.
- `nextInt()` used with no `Random` object made.
- handling exceptions with try/catch. The only exception that can be thrown is a run-time exception. The idea of run-time errors is that they come about because of mistakes made by the developer and should be corrected by writing correct code.
- using 6 instead of `answers.length` – not wrong as such, but it is always better to be general so that changes can easily be made. In this case, using `answers.length` would mean that extra elements could be added to the answers array without needing to change any other part of the class.

Of the candidates attempting this question 30 per cent had no idea how to write a sensible answer and therefore received no credit.

Part b

i. 55 per cent of candidates understood that the purpose of `System.in` was to associate the `Scanner` object with the keyboard. Of those answering incorrectly, 29 per cent thought that the purpose was to tell the Scanner class to accept input from an object called 'in'; 12 per cent answered 'None of the above'; and 4 per cent thought that the purpose of `System.in` was to link the `Scanner` object with the output of the Screen.

ii. 75 per cent knew the correct answer was that the program would end with an exception; while 20 per cent wrongly thought that the program would output `"What is your name?"`; and the remaining 5 per cent chose '(D) None of the above.'

iii. 67 per cent of candidates correctly gave (A) as the answer, with (D) being the wrong answer chosen by 25 per cent, and the remaining candidates choosing (B). The reason that (A) is correct is that since the *nextInt()* method did not read the new line after the number 5 was entered, it was read by the `nextLine()` method and stored in the *name* variable. Hence, Fred did not get a chance to enter his name, as the program thought that it already had his name.

Part c

The correct answer to (c)(i) was:

```
[ ]
####
########
[ ]
*
[ ]
****
****
[ ]
**
**
**
**
//in the above [ ] indicates a blank line.
```

Candidates did not receive full credit unless they indicated where the blank lines were. Nearly 70 per cent achieved full credit for this question, with half of the remainder receiving some credit for partially correct answers; such as answers with blank lines missing; or with only the first five lines correctly given.

Most candidates (79 per cent) understood that the answer to part (ii) was (C) – the *print()* method was overloaded. A few thought that the program would not compile because of the two *print()* methods, with a similar number thinking that the two *print()* methods were an example of method overriding.

Question 4

This question was attempted by 63 per cent of candidates.

Part a

In part (i) candidates understood that the contents of the file *filey.java* would appear on the screen (or be written to standard output). There were no incorrect answers.

In part (ii) 97 per cent of candidates understood that -1 signifies that the end of the file has been reached with the remainder seeming to think that –1 would be output.

In part (iii) most candidates (84 per cent) understood that nothing would happen, or nothing would appear on the screen and explained that this was because the condition `t!=-1` would be false from the start, as the end of the file would be found by the *filey* program immediately. 3 per cent of candidates lost credit by not explaining why there would be no output. In total 13 per cent of candidates lost all credit by answering that there would be an exception or an error.

In part (iv) 90 per cent of candidates understood that the program would stop with an uncaught exception (`FileNotFoundException`), with the remainder gaining no credit for answering that there would be no output. No deduction was made from those candidates who gave the wrong name for the exception thrown, since the question did not ask for this.

Part b

A good answer to part b would have said that the *swap(int[], int, int)* method has been written to swap elements using their index numbers, and so will treat the `int` parameters 56 and 10 given in it as index numbers for the *myNos* array. Since the array has 8 elements, it is indexed from 0 to 7 inclusive, and hence there will be a run-time error when the method attempts to access array item 56, an array index that is out of bounds.

This question was answered correctly by 68 per cent of candidates. There were no common incorrect answers, each one was unique, indicating that candidates were badly prepared for the examination and were guessing. For example, reasons given for the run-time error included:

- because the swap method is not properly defined
- has called method myNos, where method has an array of ints but swap method can only take two elements
- swap is swapping fixed length arrays, cannot be used for arraylists
- myNos global variable cannot be used as a temp variable in swap, myNos need to be empty to take values when swapping
- array myNos cannot be passed to swap
- array class not imported or declared
- the swap method is void.

Part c

Around half of candidates, 46 per cent, gained full credit for this question. Again, candidates either understood and answered correctly, or various unique wrong answers were seen from badly prepared candidates relying on guesswork.

A good answer would have said that if the search term is found then the *find(int[], int)* method successfully returns its index number. The default value returned if the search term is not found is zero. Since arrays are subscripted from zero there is ambiguity if zero is returned – does this mean that the search term was not found, or does it mean that the search term is the first item in the array? In either case, the first item in the array will be replaced with a new term, meaning that in every case where the search term is not found, the first item in the array will wrongly be replaced with a new term. This will not cause any compilation or run-time errors.

Part d

Most candidates gained some credit for their answer, but only 48 per cent gained full credit, with 13 per cent gaining no credit at all since they clearly had no idea how to approach the question.

The following is an example of an answer that would gain full credit; other correct answers are possible.

```
import java.io.*;
public class swapabxy{

   public static void cat(String s) throws Exception{
      BufferedReader inone=new BufferedReader(new
      FileReader(s));
      int t=inone.read();
      while (t!=-1){
         if (t ==’a’)System.out.print(‘x’);
         else if (t==’b’) System.out.print(‘y’);
         else System.out.print((char)t);
         t=inone.read();
      }
   }

   public static void main(String[] args) throws
   Exception{
   cat(args[0]);

}
```

Common errors seen were:

- Using double quotes around `char` variables; or not putting `char` variables in any quotes, single or double, hence giving a compilation error.
- Not updating the variable used to read from the file, *t* in the above example, in the `while` loop meaning: (1) that the loop would be infinite once entered; and (2) only one character would ever be read and output.
- not printing the (unswapped) contents of the file, but only printing the swapped characters.

Question 5

This question was attempted by 51 per cent of candidates.

Part a

i. 87 per cent of candidates understood that the output would be `gamma` only; although a few wrongly thought that the output would be `beta`, or `beta` followed by `gamma`. Another mistake was answering that the class should have had ‘throws Exception’ included in the heading of the main method; namely:

```
public static void main(String[] args) throws
Exception{
```

i. Most candidates understood that since the main method was catching `Exception`, the super type of all exceptions, that meant that all possible exceptions would be caught, not thrown.

i. 83 per cent of candidates understood that the output would be `Proceed`, with the remainder answering incorrectly that it would be `divide by zero`.

Part b

i. The correct answer was that the first method was implementing **binary search** and the second **linear search**. Most candidates answered correctly; some candidates described linear search as a 'brute force' method, and the examiners accepted this answer. A few candidates (10 per cent) muddled the name of one of the algorithms but got the other right; for example, calling binary search 'boolean search' or 'bubble sort'. 27 per cent received no credit, giving answers that were entirely wrong, such as 'the methods are finding the length of an array' or 'the algorithm is recursion'.

ii. To gain full credit for part (ii) candidates needed to state that they would use the *Easy(int[], int)* method as it implements binary search, which is faster than linear search. Ideally, candidates would go on to say that binary search relies on a sorted method and can give wrong answers when given unsorted arrays, hence *Trex(int[], int)* would be the best method to use with an unsorted array; 30 per cent of candidates made this point. Unsurprisingly, the candidates who received no credit for part (i) could not give a sensible answer to this part, writing such things as 'I would choose Easy as it is easier to read'; or 'I would choose the second one as it is less complicated.'

Part c

50 per cent of candidates gained full marks for this question, 20 per cent gained partial credit, and the remainder appeared not to have any idea how to attempt an answer; for example, writing a method with a for loop that outputs two spaces at each iteration; or asking the user for input, reading the input then ending the method.

The following method answers the question, including ensuring that there are no spaces added after the final character:

```
public static void insertSpaces(String word) {
   int length = word.length() - 1;
   String s = "";
   char c;
   for (int i = 0; i < length; i++) {
      c = word.charAt(i);
      s += c + "  ";
   }
   //do not add a space after the last character
   c = word.charAt(length);
   s += c;
   System.out.println(s);
}
```

The most common mistake was outputting two spaces after the final character, sometimes because no attempt was made not to print spaces after the final character; and sometimes because an attempt was made but the logic failed. Other errors related to the attempt to prevent the method from outputting spaces after the final character; for example, failing to print the final character at all, and printing the final character twice; the first time with two spaces after it, and the second time without spaces.

i. The best answers checked that the `String` given to the method was not empty or `null`. One good answer returned the statement 'Empty String' if given a zero length or null String.

Question 6

This question was attempted by 33 per cent of candidates.

Part a

Correct answers to the true or false questions posed were:

a. TRUE [75 per cent answered correctly]

b. FALSE [19 per cent answered correctly]

c. TRUE [75 per cent answered correctly]

d. TRUE [69 per cent answered correctly]

e. FALSE [75 per cent answered correctly]

f. TRUE [75 per cent answered correctly]

g. TRUE [75 per cent answered correctly]

h. TRUE [69 per cent answered correctly]

The most troublesome answer for candidates was (B) where only about 1 in every 5 candidates attempting the question understood that static methods **cannot** access instance variables.

Part b

In parts (i) and (ii) only 60 per cent of candidates attempting the questions achieved full marks. For (i) constructor with two parameters should look similar to the following:

```
public StringAndInt (String y, int x){
   s = y;
   i = x;
}
```

Some candidates lost credit by writing an empty constructor; namely:

public StringAndInt (String y, int x){}.

Constructors are used to initialise the instance variables of an object; therefore candidates were expected to use their constructor to assign values to the instance variables *s* and *i*. Other candidates lost credit by not knowing how to write the heading of a constructor, writing such things as:

- constructor1(String s,int i)
- public apple(String a, String b)
- public class StringAndInt

In part (ii) some candidates again lost credit by writing an empty constructor, despite being asked to use the no-argument constructor to initialise the instance variables to “goodbye” and 5.

Again, candidates struggled to write a sensible heading, writing such things as:

- constructor2
- public static void
- public class StringAndInt

It was striking that the candidates who could not write a proper heading for their second constructor also thought that as they were told not to give the constructor parameters, they did not have to include the round brackets in the constructor’s heading. These candidates had clearly not spent enough time attempting to write their own classes, and doing the programming exercises in the subject guide.

One correct answer to part (ii) would be:

```
public StringAndInt(){
   s = "goodbye";
   i = 5;
}
```

A simple answer to part (iii) to achieve full credit would be:

```
public String toString(){
   return "(" + s + "," + i + ")";
}
```

Other correct answers were possible.

For this question, the majority of candidates, 75 per cent, knew very well how to answer, while the remaining 25 per cent had no idea and wrote a variety of incorrect methods and classes.

Part c

In part (c) only 6 per cent of candidates attempting this question achieved full credit. In fact, the majority, 56 per cent, received no credit at all, with the remaining 38 per cent receiving partial credit.

A simple, correct, answer would be:

```
public class Bigger extends CompareThis {
   public int compare(int x, int y) {
      if (y < x) return 1;
      if (x < y) return -1;
      return 0;
   }
}//other (similar) correct answers were possible
```

Mistakes seen in answers that achieved some credit included:

- Making the *compare(int, int)* method a `static int` method.
- The *compare(int, int)* method is *abstract*, but implemented.
- The *compare(int, int)* method included the word 'implement' in its heading.

Examples of answers that received no credit include:

- The heading of the method reads

  ```
  implement int compare (int x, int y),
  ```

  and the method attempts to parse *x* and *y* from `args[0]` and `args[1]` respectively.
- The *Bigger* class does not extend the *CompareThis* class and the statements in the method read:

  ```
  int x = 5;
  int y = 10;
  compareThis(x&&y);
  return x+y;
  ```
- The *abstract* key word is included in the *Bigger* class declaration, the method has the wrong name, and there is one statement in the method reading:

  ```
  if (x = = compare(x,y) return 1
  ```

- The *compareThis* class is directly implemented as follows:

```
public abstract class compareThis{
   abstract int compare(int x, int y);
   int x = Math.max(compare(int, int);
   int s; = s.compareTo(int, int);
   for (int i =0; i<Y;i++);
}
```

From the above lists, it can be gathered that candidates had moderate to severe trouble understanding how to extend and implement the *CompareThis* class and its *compare(int, int)* method.

Conclusion

The examination was attempted well by the majority of candidates, with some candidates showing an excellent grasp of Java that went well beyond the basics. A significant minority of candidates could have done much better given the sort of familiarity with basic programming concepts that can only be achieved by attempting to write and compile statements, methods and classes.

Examiners' commentaries 2016–17

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

General remarks

The majority of candidates did very well on this examination. However, a significant minority would be advised to ensure much better familiarity with basic programming concepts that can only be achieved by attempting to write and compile statements, methods and classes. Therefore it is important to practise, practise, practise.

Comments on specific questions

Question 1

This was a popular question, chosen by 98 per cent of candidates, with many getting full credit.

Part a

i. 83 per cent of candidates knew that the class would compile and output `false`, while 12 per cent thought the class would not compile, and 5 per cent thought that the class would compile and output `true`.

ii. 98 per cent of candidates understood that the *Bool3* class would output `hello`. The remainder thought that the class would not compile.

iii. 82 per cent of candidates wrote a correct answer. Different correct answers were possible, such as a class with an empty main, or a main method that declared and initialised a variable. Errors seen included writing a main method only and writing a class without a main method (a class with no main will not run, and candidates were asked to write a class that would compile and run). Other errors were bracketing errors such as a main method without opening and closing brackets, and writing a class with a `Scanner` object, and asking the user for input. Asking the user for input is output, so these candidates received no credit. A small number of candidates made no attempt at this part.

Part b

Part (b) was often completely right; those with mistakes were normally off by one error and were mistaken in thinking that a finite loop was infinite. Correct answers and common mistakes (in brackets) follow:

i. 9 (8) [98 per cent answered correctly]

ii. 4 (infinite, 7) [97 per cent answered correctly]

iii. 6 [100 per cent answered correctly]

iv. no output (5, 7, 2, 'error') [80 per cent answered correctly]

v. INFINITE (1) [88 per cent answered correctly]

vi. 20 (379, 40, 19) [95 per cent answered correctly]

vii. 5 (4, 8, 9) [95 per cent answered correctly]

viii. INFINITE ('no output', 8) [95 per cent answered correctly]

ix. no output (2, infinite, 'error') [82 per cent answered correctly]

Part c

The output asked for should have been as follows:

```
*
**
***
****
#
#
#
****
***
**
*
```

The majority of answers were completely right. The most common error was putting the #s in a horizontal line, with the rest of the output given correctly. A few got the triangle shapes given by the stars the wrong way round; namely:

```
****
 ***
  **
   *
```

A very few candidates gave output that was completely wrong; these candidates were clearly guessing and were very badly prepared for the examination.

Question 2

Although not quite as popular as Question 1, this question was attempted by about 95 per cent of candidates, and was done well on the whole.

Part a

Part (i) was answered correctly by all; that is, the value the program is attempting to assign to `int y`, is not an integer, causing a compilation error.

For part (ii) various incorrect answers were given, such as the variable *s* was out of scope in the main method, or *s* should have an access modifier, or *s* should be declared in the main method since declaring a variable outside the main method was a syntax error. Full credit was gained by 67 per cent of candidates by writing that since *s* was not a static variable, it could not be accessed in a static context, or an equivalent answer such as since *s* was an instance variable it could not be directly accessed by main, or that *s* could only be accessed by main through making and using an object of the class.

In part (iii) all candidates gained full credit, answering that there should not be double quotes around '1', or, equivalently, that the class was attempting to assign a `String` to an `int` variable. Unfortunately, due to a printing error on the question paper, class *C* was missing its closing bracket, so some candidates gave that as the error, also gaining full credit.

Part (iv) was answered correctly by 84 per cent of candidates. A good answer would have said that the *add(int, int)* method had no return type. An even better answer would have said that it should have the `int` return type.

Some candidates were clearly guessing, giving answers such as: there are no errors; int k has not been initialised; add is a static method; method add is repeated twice; parameters do not match.

Part b

In part (b) the correct answers were:

i. NO [96 per cent answered correctly]

ii. YES [100 per cent answered correctly]

iii. YES [99 per cent answered correctly]

iv. YES [98 per cent answered correctly]

v. YES [98 per cent answered correctly]

vi. YES [20 per cent answered correctly]

vii. YES [95 per cent answered correctly]

viii. YES [75 per cent answered correctly]

ix. NO [76 per cent answered correctly]

Four per cent of candidates correctly noted that (vi) would type check, but would give a run time error due to the out of bounds number given to the *charAt(int)* method.

Part c

For part (c) many candidates wrote answers similar to those given below that were logically and syntactically correct:

i.

```
static int max (int x, int y){
   if (x>y) return x;
   else return y;
}
```

ii.

```
static int maxOf3 (int x, int y, int z){
   int a = max(x,y);
   int b = max(a,z);
   return b;
}
```

Other correct answers were possible, and a variety of different logical approaches were seen, particularly to part (ii).

In their answer to part (i) some candidates did not cover the case of x being equal to y, as follows:

```
static int max (int x, int y){
   if (x>y) return x;
   else if (y>x) return y;
}
```

In this case what happens when x equals y is undefined, and the compiler will flag this as a missing return statement. Since this mistake means that the method would not compile, these candidates gained no credit.

The most common reason for losing credit for part (ii) was when candidates did not use the method written in part (i) in their answer (no credit). Another common mistake seen in both parts was in not giving one or both of the methods a return type (some credit).

Another reason for losing credit was when candidates did not understand how to work with the parameters of their methods, and instead used local parameters, or assigned the parameter to a local parameter and then worked with the local parameter. This showed a lack of practice in hands-on programming tasks, and generally poor examination preparation.

Question 3

This question was attempted by 44 per cent of candidates.

Part a

The examiners were looking for an answer similar to the following:

```
public static String magic8Ball(){
   Random r = new Random();
   int index = r.nextInt(answers.length);
   return answers[index];
}
```

No credit was given for first checking to see that the answers array was not empty, although this would have been a sensible thing to do. One candidate implemented this check, returning an empty `String` if the array had no entries.

The following mistakes were seen in otherwise correct methods (candidates received all or some of the credit):

- using answers.length-1 to make the random number. This would give a number between 0 and 5 inclusive, meaning that the final array element could never be chosen.
- the method was written with an `array` parameter, despite being called with no parameter in the main method given with the question.
- logically correct method with incorrect syntax for the `Random` variable.
- using 6 instead of answers.length – not wrong as such, but it is always better to be general so that changes can easily be made. In this case, using answers.length would mean that extra elements could be added to the answers array without needing to change any other part of the class.

Part b

i. 78 per cent of candidates understood that the purpose of `System.in` was to associate the `Scanner` object with the keyboard. The most popular incorrect answer was (A), given by 18 per cent of candidates with the remainder giving (D) as their answer.

ii. 74 per cent knew the correct answer was that the program would end with an exception, while 22 per cent wrongly thought that the program would hang, waiting for input, and the remaining candidates thought that the program would output `"What is your name?"`

In part (iii) 74 per cent of candidates correctly gave (A) as the answer, with (D) chosen by 22 per cent of candidates, followed by (B). The reason that (A) is correct is that since the *nextInt()* method did not read the new line after the number 5 was entered, it was read by the `nextLine()` method and stored in the *name* variable. Hence, Fred did not get a chance to enter his name, as the program thought that it already had his name.

Part c

Candidates did not receive full credit unless they indicated where the blank lines were. The correct answer to (i) was:

```
[ ]
##
###
[ ]
*
[ ]
**
**
**
[ ]
***
***


//in the above [ ] indicates a blank line.
```

67 per cent achieved full credit for this question, with most of the remainder receiving some credit for partially correct answers, such as answers with blank lines missing, or with only the first five lines correctly given. 7 per cent of answers seen were completely wrong and received no credit.

Most candidates understood that the answer to part (ii) was (B): the *print()* method was overloaded. 11 per cent wrongly answered that the two *print()* methods were an example of method overriding, and 3 per cent chose '(D) None of the above' as their answer.

Question 4

This question was attempted by 51 per cent of candidates.

Part a

In part (i) candidates understood that the contents of the file *filey.java* would appear on the screen (or be written to standard output). There were no wrong answers, but some incomplete ones where candidates stated that the contents of the file given by `args[0]` or the contents of the file given by *s* would be printed; candidates were explicitly asked to state what would happen if the file *filey.java* was given as input.

In part (ii) all candidates understood that -1 signifies that the end of the file has been reached.

In part (iii) most candidates understood that nothing would happen, or nothing would appear on the screen, but many could not successfully explain that this was because the condition `t!=-1` would be false from the start, as the end of the file would be found by the *filey* program immediately. A few candidates incorrectly thought that the result of running the *filey* class with an empty input file would be that the program would stop with a run time error.

In part (iv) all candidates understood that the program would stop with an uncaught exception (`FileNotFoundException`).

Part b

A good answer to part b would have said that running the *SwapInt* class produced a run-time error because the method call in the main method is trying to swap two array elements by directly referencing them, rather than using their index numbers. The *swap(int[], int, int)* method has been written to swap elements using their index numbers, and so will treat the `int` parameters 99 and 16 as index numbers for the *myNos* array. Since the array has 6 elements, it is indexed from 0 to 5 inclusive, and hence there will be a run-time error when the method attempts to access 99, an array index that is out of bounds.

This question was answered correctly by the majority of candidates, with no pattern to the various incorrect answers seen, where candidates were clearly badly prepared for the examination and were guessing. For example, wrong answers included: the method swaps two numbers but three objects are passed; cannot swap values in a static array; and the main method cannot access the array.

Part c

Most candidates, 71 per cent, gained full credit for this question. Again, candidates either understood and answered correctly, or various wrong answers were seen from badly prepared candidates who were guessing.

A good answer would have said that if the search term is found then the *find(int[], int)* method successfully returns its index number. The default value returned if the search term is not found is -1. The *replace(int[], int, int)* method does not check to see whether or not a valid index number has been returned, and thus in every case where the search term is not found, the method will attempt to access the array element subscripted by -1, which does not exist. This means that when the search term is found the program works correctly, and when the search term is not found the program throws a run-time error.

The best answers seen suggested ways of fixing the problem, by having the *replace(int[], int, int)* method check the value of the *temp* variable after calling the *find(int, int)* method, and printing an appropriate error message if the value stored in *temp* is not in range.

Part d

Most candidates gained some credit for their answer, but only 48 per cent gained full credit. Excellent answers included those that closed the file after the end of the while loop (not necessary for full credit but good practice) and those that took care to swap both lower case and upper case letters; namely, 'c' for 'd' and 'C' for 'D', also 'd' for 'c' and 'D' for 'C'. The majority of candidates only swapped lower case letters, which gained them full credit when appropriate.

Correct answers seen included those that put all their statements into the main method, and those that used a method. Both answers could achieve full credit, although using a method is better programming practice. The following is an example of a correct answer using a method to read from the file, swap characters as necessary and print the output:

```
import java.io.*;
public class swapcd{

   public static void cat(String s) throws Exception{
      BufferedReader inone =new BufferedReader(new
      FileReader(s));
      int t=inone.read();
      while (t!=-1){
         if (t =='c')System.out.print('d');
         else if (t=='d') System.out.print('c');
         else System.out.print((char)t);
         t=inone.read();
      }
   }

   public static void main(String[] args) throws
   Exception{
   cat(args[0]);
   }
}
```

Most errors seen were unique to the candidate, such as where the logic of the method was correct, but the syntax would give a compilation error. One common error was failing to read from the file inside the `while` loop, meaning that once the `while` loop was entered it could not terminate, as the stopping condition would not become true.

Question 5

This question was attempted by 85 per cent of candidates.

Part a

i. Most candidates understood that the output would be `division done` only, although 2 per cent wrongly thought that the output would be `division done` followed by `not a number`.

ii. Most candidates understood that the output would be `zeta`, although 2 per cent incorrectly thought it would be `epsilon`.

Part b

i. The examiners were looking for a simple answer, that the methods had both been written in order to search an array. Most candidates answered correctly, although a few answers showed some misunderstanding by stating that the methods were looking for the position of an item in an array, when in fact the methods both returned `true` if the item was found, and `false` if it was not. Neither method would return a position for any item found in the array.

ii. To gain full credit for part (ii) candidates needed to state that they would use the *Easy(int[], int)* method as it implemented binary search, which is faster than linear search as implemented by the other method. Ideally candidates would go on to say that binary search relies on a sorted method and can give wrong answers when given unsorted arrays, hence *Trex(int[], int)* would be the best method to use with an unsorted array, or indeed with an array sorted in descending order, as *Easy(int[], int)* assumed that the array was sorted in ascending order.

A few candidates thought that the first method implemented insertion sort; some gave the sorting method as 'boolean'; although all understood that the second method implemented linear sort. Another common error was stating that the *Trex(int[], int)* method would be better as it was 'simpler' or 'easier to read'. Another error was thinking that the *Easy(int[]*, int) method returned true or false but the *Trex(int[], int)* method returned an index of location and hence would be the better method to use.

Part c

83 per cent of candidates gained some marks for this question, with the remainder not really having any idea how to attempt an answer; for example, an answer whose logic would print each character on its own line, but would not compile because of numerous errors in the syntax; or an answer that added two spaces to the end of the String parameter, but only if it equalled 'true'.

The following method answers the question, including ensuring that there are no spaces added after the final character:

```
public static void insertSpaces(String word) {
   int length = word.length() - 1;
   String s = "";
   char c;
   for (int i = 0; i < length; i++) {
      c = word.charAt(i);
      s += c + "  ";
   }
   //do not add a space after the last character
   c = word.charAt(length);
   s += c;
   System.out.println(s);
}
```

In the above example the *length* variable is equal to the length of the `String` parameter, minus one. Hence it can safely be used to access the final character of the String without causing an out of bounds error. Many candidates used exactly the structure given above in their answers; that is, they attempted to print the final character of the `String` outside of the loop in order not to add spaces after it. The most common error these candidates made was trying to access the character given by the length of the `String`. This meant that, since characters in Strings are subscripted from 0 to the length of the `String` minus one, there would be a run-time error (`StringIndexOutOfBoundsException`).

The best answers seen checked that the `String` given to the method was not empty or `null`, and checked the word length too, only starting the loop if the length was greater than 1.

Question 6

This question was attempted by 26 per cent of candidates.

Part a

Correct answers to the true or false questions posed were:

a. TRUE [46 per cent answered correctly]

b. TRUE [81 per cent answered correctly]

c. TRUE [50 per cent answered correctly]

d. TRUE [81 per cent answered correctly]

e. TRUE [87 per cent answered correctly]

f. FALSE [100 per cent answered correctly]

g. TRUE 94 per cent answered correctly]

h. TRUE 62 per cent answered correctly]

Part b

A constructor with two parameters should look similar to the following:

```
public IntAndString(int x, String y){
   i = x;
   s = y;
}
```

While most candidates gained full credit for this question, some lost credit by writing an empty constructor. Constructors are used to initialise the instance variables of an object; therefore candidates were expected to use their constructor to assign values to the instance variables *i* and *s*. Other candidates lost credit by not knowing how to write the heading of a constructor, writing such things as:

- public static void IntAndString(int i, String s)
- public class IntAndString(int i, String s)
- public class(int i, String s)

Only half of the candidates attempting (ii) achieved full credit; 12 per cent received partial credit; and the rest none.

Candidates again lost credit by writing an empty constructor; that is, simply writing down:

```
public IntAndString(){}
```

And again candidates struggled to write a sensible heading, writing such things as:

- `public class IntAndString2 extends IntAndString`
- `public class SecondConstruct()`
- `public class()`

Some candidates were completely baffled by the instruction to write a second constructor, despite all candidates knowing in answer to part (a) that a class can have more than one constructor. These candidates wrote answers such as the following:

- `IntAndString test1 = new IntAndString(1, "hello");`
- `public IntAndString{new IntAndString(1,"hello");}`

The correct answer to part (ii) would be:

```
public IntAndString(){
   i = 1;
   s = "hello";
}
```

A simple answer to part (iii) that would have achieved full credit would be:

```
public String toString(){
   return "(" + i + "," + s + ")";
}
```

Other correct answers were possible.

Many errors were seen, with only a quarter of candidates gaining full marks for this part. Errors included:

- Writing a void method that output the values of the instance variables using `System.out.println()`.
- Writing a static method.
- Not having the `String` type in the heading (or any type at all).
- Writing a `String` method with parameters.
- Attempting to directly return the instance variables without making a `String`.

Part c

In part (c) only 38 per cent of candidates attempting the question achieved full credit. Of the rest 44 per cent achieved partial credit and 18 per cent none at all.

A simple, correct answer would have been:

```
public class Smaller extends CompareThis {
   public int compare(int x, int y) {
      if (y > x) return 1;
      if (x > y) return -1;
      return 0;
   }
}//other (similar) correct answers are possible
```

Mistakes seen included:

- Making the *compare(int, int)* method a static method.
- Making the *compare(int, int)* method a static void method.
- The *compare(int, int)* method is abstract, but implemented.
- The *Smaller* class and the *compare(int, int)* method are both abstract.
- Implementing the *compare(int, int)* method in the *CompareThis* class (that is, not writing a *Smaller* class to extend *CompareThis*.
- Writing a main method to do the work of the *compare(int, int)* method.
- Doing the work of the *compare(int, int)* method in the class, but not in a method.

From the above list of errors it can be gathered that many candidates had severe trouble understanding how to implement the *CompareThis* class and its *compare(int, int)* method.

Conclusion

This examination was attempted well by the majority of candidates, with some candidates showing an excellent grasp of Java that went well beyond the basics. A significant minority of candidates could have done much better had they been familiar with basic programming concepts. This familiarity is achieved through practice – and in particular by attempting to write and compile statements, methods and classes.