Examiners’ commentaries 2015–2016

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

Comments on specific questions

Question 1

For part (a) candidates gave the expected answers most of the time (`boolean` or `byte`; `long`; `int`; and `double`) but some candidates did not appreciate that primitive data types start with lower case letters in Java, for example writing `Boolean` instead of `boolean`. A minority of candidates thought that the lower case ‘l’ was an upper case i and put Integer – showing again that some candidates did not understand that in Java, primitive types start with lower case letters. A very common error was writing Integer, instead of `int`; this mistake was made by many candidates who otherwise gave correct answers.

For part (b) the examiners accepted different answers provided that they were reasonable, as follows:

```
(i) int / short / long
(ii) double / float / int / short / long
(iii) boolean
(iv) char / boolean
```

In part (c) a minority of candidates lost marks by writing a program that would compile but not run because it had no main. A variety of answers were given, including classes with a statement in main that declared a variable, or classes with an empty main method but with instance variables and methods. The simplest answer was:

```
public class CA{
   public static void main(String[] args){}
}
```

Part (d) could be answered very simply as follows:

```
public class GetArguments{
   public static void main(String[] args){
      System.out.println(args.length);
   }
}
```

A few candidates wrote a main method without a class declaration – candidates were asked to write a program –not simply a method. Avery puzzling mistake was printing `args.length-1` rather than `args.length`. There were also a large number of candidates who wrote programs involving a `for` loop. This included counting the number of command line arguments by cycling through them in a for loop, with the number of iterations determined by the size of the `args[]` array, `args.length`. Other candidates printed the number of command line arguments in a `for` loop; which was also problematic since the number does not change the choice of a loop to display it more than once. A common mistake was printing all of the different command line arguments in a loop; the examiners assumed that such answers were given by candidates who appeared to havemisread the question.

Part (e) was mostly answered incorrectly, with many candidates producing classes with multiple compilation errors. The most common mistake made by candidates was to put statements that belonged in the *toString()* method into the *isDone()* method. Only a minority of candidates gave a completely correct answer as follows:

```
public class ToDoList{
   private String item;
   private boolean done;
   public ToDoList(String item, boolean done){
      this.item = item;
      this.done = done;
   }
   public boolean isDone(){
      return done;
   }
   public String toString(){
      String s = ("The to do list item: "+item+"\n");
      if (isDone()) s = s +  "has been done";
      else s = s +"has not been done";
      return s;
   }
   public static void main(String[] args){
      ToDoList toDoList = new ToDoList("take cat to vet",
      false);
      System.out.println(toDoList);
   }
}
//note that the order of the instance methods is arbitrary
```

Question 2

In part (a) candidates achieved the majority of the marks. In particular, the first four answers were given correctly by all candidates. Correct answers were:

```
(i)   YES
(ii)  YES
(iii) NO
(iv)  YES
(v)   NO
(vi)  YES
(vii) YES
```

In part (b) most candidates failed to appreciate that the calculation would be 16 divided by 9, since 16 is the length of the `String`, and the result of this would be 1 because integer division gives the number of divisions and throws the remainder away. Since 9 divides into 16 one time with 7 left over, the result of integer division is 1. The answer therefore is the character at position 1 in "`Scorpius`", which is `'c'` (A).

In part (c) most candidates got their answers almost completely correct, as follows:

```
(i)   10 STARS
(ii)  5 STARS
(iii) infinite loop (some candidates equivalently wrote that
the loop would continue 231-1 times, i.e. up to the limit on
int in Java).
(iv)  30 stars
(v)   infinite loop
(vi)  no output
(vii) for(int i=0;i<5;i++); System.out.println("**");
//semi-colon ends for loop, two stars printed.
```

No correct answers given.

```
(viii)      10 stars
```

In part (d) a few candidates thought that they could answer the question with `Math.max(arr.length)` or `Math.max(arr[])`.The first answer is quite a serious misunderstanding of programming fundamentals. The second suggests confusion about what parameters the *Math.max()* method will take; it is overloaded to take two doubles, ints, longs or floats, it will not take an array.

The most common mistake was candidates forgetting to make sure that given an empty or null array the method would return zero. Some candidates remembered this but in such a way that a run-time error would be given by a null array, as the check on the array's size was only done after attempting to access an array element. The worst mistake was that some candidates made an array local to their method, and then searched this empty array. This was a serious misunderstanding of the basics of programming using methods. Some candidates made a local array, then added some values to it in order to be able to search it; perhaps the candidates would have written better answers had they reflected on the utility of such a method.

Many candidates wrote an answer similar to that given below, which was logically and syntactically correct.

```
static int longestLength (String [] arr){
   int longest=0;
   for (int i=0;i<arr.length;i++){
      if (arr[i].length()>longest) longest=arr[i].length();
   }
   return longest;
}
```

The method does not explicitly deal with the case of an empty array, however, if given a null array, the method will not go into the for loop, hence it will not try to access array elements that do not exist and will return zero. Other candidates wrote methods that explicitly dealt with the case of the empty array (e.g. starting their methods with `if (arr.length == 0) return 0`;) and this approach was also satisfactory

Question 3

Part (a)(i) asked candidates to explain an error in a small program. The error was described in various ways, such as `String` cannot be converted to `int`, or the statement `s = 1`; should be `s = "1"`;. All explanations showed understanding that the `String` variable was having a number assigned to it and full credit was given.

In part (a)(ii) all candidates successfully identified the error that the variable *t* was being defined twice within the same scope. Again this was explained in different ways (e.g. *t* cannot be a `String` and an `int`) but provided the answer demonstrated an understanding of the issue, full credit was given.

In part (a)(iii), the most common error made was stating (correctly) that class A would not compile, without giving any reason why. Some candidates said that the problem was that the variable *z* was declared outside the main method, but for full credit candidates needed to write that as *z* was an instance variable, it could not be accessed directly by main (i.e. it could not be accessed in a static context). A minority erroneously thought that class C would not compile, either because *z* should not be a static `int`, or because it would not be possible to change a static `int` in the main method.

Part (b) was answered mostly incorrectly on the whole.. In part (b)(i) either

```
if ((t1 > t2) && (b==true)) t1=t2+1;
```

OR

```
if ((t1 > t2) && (b)) t1=t2+1;
```

were marked as correct, when seen, which was not often. Some candidates included in their answer that the value of *b* was determined by: `b = t1 > t2`; but there was no evidence in the statements given that *b* depended on *t1* and *t2*, so no credit was given. Many candidates wrote an unnecessary `while` loop, which often introduced behaviour not in the original code fragment. Some candidates wrote a while loop that would break once the desired behaviour had been achieved (e.g. `if (t1 > t2) while (b) {t1=t2+1; break}`).This meant that the statements would behave the same as the original. However, the loop was an unnecessary complication, since the point of a loop is to repeat behaviour.

Part (b)(ii) produced more incorrect answers than (i). Candidates could, for full credit, have answered:

```
if (t==5) s = "hellogoodbye";
System.out.println(s);
```

`//S.o.p()` not necessary for full credit

In general most candidates failed to appreciate that the `if(!(t==5)) s = s + "goodbye";`statement was not necessary in their answer; these candidates got no credit. All candidates who removed this statement got at least some credit, although completely correct answers were rare. The most common answer was:

```
if(t==5) s = "hello"+"goodbye";
if(!(t==5)) s = s + "goodbye";
System.out.println(s);
```

In part (b)(iii), most candidates could successfully rewrite the `for` loop as a `while` loop, however a common mistake is shown below:

```
public static int CoinToss2(int n){
   int sum = 0;
   int i = 0;
   //i must be declared and initialised here
   while (i < n){ //putting while (int i < n) mistake!
      int r = (int)(Math.random()*2);
      sum = sum + r;
      i++;
   }
   return sum;
}
```

Some candidates wrote a correct while loop, counting down from the value of the *n* variable to zero, as follows:

```
public static int coinToss(int n){
   int sum = 0;
   while (n>0){
      int r = (int)(Math.random()*2);
      sum = sum + r;
      n--;
   }
   return sum;
}
```

In part (c) a very few answers were seen with rows missing, or where the shape given by the stars was completely wrong. Most candidates gave completely correct answers as follows:

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

Question 4

In part (a)(i) most candidates understood that the output would be the contents of file filey.java, which would appear on the screen (or be written to standard output).

In part (a)(ii), most understood that -1 signifies that the end of the file has been reached.

Most understood in part (a)(iii) 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.

All candidates knew that the answer to part (b) was 'h', and most successfully answered (c) with:

`s.charAt(1)`

`s.charAt(s.length() - 1)` One common mistake was to write `s.charAt(-1);` for the second answer.

For part (d) a good answer and the correct answer most often given was similar to:

```
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]);
   }
}
```

Some mistakes were seen, most of them minor, including:

- characters were swapped, but nothing was output
- swapped all ‘a’s for ‘x’s but not all the ‘b’s for ‘y’s, or *vice versa*
- did not take a file name from the command line (i.e. the equivalent of `cat(args[0]);` in the above was missing)
- `t=inone.read();` missing or in the wrong place (i.e. not in the while loop)

The most serious mistake was poor logic (e.g. the program would print an ‘x’ instead of an ‘a’ and a ‘y’ instead of a ‘b’ but then wouldprint the ‘a’ or ‘b’). This was normally because there were no `else` statements, i.e. the swapping statements inside the loop would be:

```
if (t ==’a’)System.out.print(‘x’);
if (t==’b’) System.out.print(‘y’);
System.out.print((char)t);
```

An original, correct, approach taken by some candidates was:

```
import java.io.*;
public class swapabxy{
   public static void main(String[] args) throws Exception{
      BufferedReader in = new BufferedReader(new FileReader(args[0]));
      int a = in.read();
      while(a !=-1){
         if(((char)a)==’a’) a=(int)’x’;
         else if (((char)a)==’b’) a=(int)’y’;
         System.out.print((char)a);
         a = in.read();
      }
   }
}
```

Question 5

In part (a) candidates showed good understanding of static methods and variables. Most understood that static methods can be run before an instance of the class is made and that static methods cannot operate on instance variables. Many appreciated that static variables hold the same value for every instance of the class and that instance variables **do not** hold the same value for every instance of the class. A large minority appreciated that static methods are often used for utility methods.

To answer (b)(i) candidates were given the following statements as a hint:

```
Random r= new Random();
int randomA = r.nextInt(6);
```

Most candidates then simply took the statements and applied them directly to their answerwith no follow-on.. The examiner had expected that at least a few candidates would understand that using the length of the tosses array as a variable (see below) would be better than using 6 to get the random value.

```
public static String aToss(){
   Random r= new Random();
   int randomA = r.nextInt(tosses.length);
   //OR int randomA = r.nextInt(6);
   return tosses[randomA];
}
```

An example of this was writing: `r.nextInt(tosses.length)` would be better than `r.nextInt(6)`, but then using the statements as a given anyway; implying that they thought the hint was an absolute instruction.

In part (b)(ii) all candidates could successfully write a main with a test statement, writing something similar to the answer below:

```
public static void main (String[] args){
     System.out.println(aToss());
}
```

Some candidates lost credit here because their method did not return a `String`, in which case writing a test statement as above would give a compilation error as the compiler will not allow an attempt to print a `void` method.

In part (c)(i) candidates were asked to write an instance method. This method was very similar to the *aToss()* method:

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

The most common mistake was including the key word 'static' in the method heading, i.e.

```
   public static String getAnswer()
```

Since static methods are not instance methods and this was the only significant difference between the two methods, these candidates received no credit.

In part (c)(ii) the most common mistake was writing a test statement that treated the *getAnswer()* method as a static method (i.e. a method that could be called without an instance of the class being made). To receive credit for their answers, candidates needed to test the method by first making a *MagicEightBall* object:

```
public static void main (String[] args){
   MagicEightBall eightB = new MagicEightBall();
   System.out.println(eightB.getAnswer());
}
```

In part (d) candidates either wrote good answers, as in the model answer below, or wrote Java statements that seemed to be chosen at random, implying they had no real idea how to approach the problem. Of those candidates who wrote good answers, the most common mistake was forgetting to make sure that two spaces were **not** inserted after the final character in the output.

```
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);
}
```

Question 6

In part (a) mostly correct answers were given, although for full credit candidates should have written each item of output on a new line, as follows:

```
0
1
2
3
4
5
6
```

A very small minority lost credit by adding a 7 to the output. Two candidates deserved extra credit for being the only ones to notice that a semi-colon was missing from the *Array1* class, giving a compilation error. Both these candidates assumed this was an error and gave the output above, although they would also have received full credit had they written that there would be no output because of the missing semi-colon.

In part (a)(ii) full credit was given to candidates who wrote that an `ArrayIndexOutOfBoundsException` would be thrown when the program was run; candidates lost some credit with imprecise answers (i.e. stating that an error would happen, or an exception would be thrown). A large minority incorrectly wrote that the program would store 2 in position 10.

In part (a)(iii), almost all candidates understood that control would pass to the `catch` block, and hence 'bonjour' would be the output.

In part (b) while a majority gave a correct answer, many different mistakes were seen. Some candidates included a type in their constructor heading, (e.g. public String Employee) a number did not include any parameters and some even declared the instance variables inside the constructor. A correct constructor would be similar to the following:

```
public Employee(String first, String last, int age, boolean permanent){
   firstName = first;
   lastName = last;
   this.age = age;
   this.permanent = permanent;
}
```

In part (c) one common error was attempting to return the reversed `String`; the method heading given included the 'void' keyword – hence the method should not be returning anything. There were some minor syntactical errors; candidates were given most of the credit provided that the logic of their method was correct. Logic errors lost candidates more credit, the most common were:

- the method made a local variable with the same name and type as the parameter. Often this parameter was not initialised (given a value) so the method was operating on an empty (null) `String` (logic error) although the method would not compile as local variables in methods must be initialised (syntax error)
- an out of bounds error would be thrown because the candidate failed to appreciate that the character at position `word.charAt(word.length())` did not exist since Strings, like arrays, are subscripted from zero
- the method would only print one character only because the reversed char was not being concatenated (added to) the other reversed chars
- at each iteration of the loop the new `String` being made consisting of reversed chars was overwritten completely, so in the end only one character was returned
- the `for` loop was going in the wrong direction (`i--` should be `i++` or *vice versa*) meaning that reversing was not happening

Many different approaches to the logic of the program were seen, although one popular approach was:

```
static void reverse(String word) {
   int length = word.length() - 1;
   String s = "";
   char c;
   for (int i = length; i >= 0; i--) {
      c = word.charAt(i);
      s += c;
   }
   System.out.println(s);
}
```

One very simple and elegant approach was:

```
static void reverse(String word) {
   for (int i = word.length(); i > 0; i--) System.out.
   print(word.charAt(i-1));
}
```

Another, very good approach, seen a few times was:

```
static void reverse(String word) {
   for (int i = 0; i < word.length(); i++)System.out.
   print(word.charAt(word.length()-i-1));
}
```

Conclusion

The exam was attempted well by the majority of candidates, with some candidates showing an excellent grasp of Java fundamentals and able to apply their knowledge in new and creative ways. There was a minority who could have done much better given more familiarity with basic programming concepts. This could have been gained by attempting the exercises in the subject guide, Volumes 1 and 2. Programming is a skill and like all skills, it improves with practice.

Examiners' Commentaries 2015–2016

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

General remarks

The exam was well attempted on the whole, with some candidates demonstrating an excellent grasp of programming basics. There was a significant minority of candidates who lost credit by making mistakes of syntax and/or logic that could possibly have been avoided had they spent more time practising writing, compiling and running programs using the exercises in the subject guide.

Comments on specific questions

Question 1

For part (a) most candidates gained full credit writing `int; char; short;` and `double`. There were three common mistakes, the first was capitalisation. Primitive data types start with lower case letters in Java, so `Double` is not a primitive data type, whereas `double` is. The second was writing `Integer` or `integer`, instead of `int` and the third was writing `String` instead of short. `String` is not a primitive data type.

Part (b) the examiners accepted a variety of answers provided that they were reasonable, as follows:

```
(i) int / short / long
(ii) double / float / int / short / long
(iii) boolean
(iv) char / boolean
```

One answer that was not accepted was `String` for (iv); to reiterate, `String` is not a primitive data type. A variety of correct answers were seen, including main methods with variables declared and initialised, or `for` loops that had no statements to run. The examiners' favourite was a class that had the statement: `String s = "no output"`; but the simplest correct answer was:

```
public class CA{
   public static void main(String[] args){}
}
```

Some very common answers seen for part (d) were:

```
public class GetArguments{
   public static void main(String[] args){
      int count = 0;
      for(int i=0; i<args.length; i++) count++;
      System.out.println(count);
   }
}
```

This was puzzling as part (d) could be answered very simply as follows:

```
public class GetArguments{
   public static void main(String[] args){
      System.out.println(args.length);
   }
}
```

Many candidates did not seem to understand that since the *`args[]`* `array` contains the command line arguments given to the method, `args.length` gives the number of elements in the *`args[]`* `array`, (i.e. the number of command line arguments). This is a failure to understand very basic Java concepts.

In part (e), many candidates could not write suitable *isDone()* or *toString()* methods, in particular the statement: `if (isDone()) s = s + "has been done"`; was included in the *isDone()* method by most candidates. Another common error was including the following statement with the instance variables:

```
ToDoList toDoList = new ToDoList("take cat to vet", false);
```

Only a minority of candidates gave a completely correct answer as follows:

```
public class ToDoList{
   private String item;
   private boolean done;
   public ToDoList(String item, boolean done){
      this.item = item;
      this.done = done;
   }
   public boolean isDone(){
      return done;
   }
   public String toString(){
      String s = ("The to do list item: "+item+"\n");
      if (isDone()) s = s +  "has been done";
      else s = s +"has not been done";
      return s;
   }
   public static void main(String[] args){
      ToDoList toDoList = new ToDoList("take cat to vet",
      false);
      System.out.println(toDoList);
   }
}
//note that the order of the instance methods is arbitrary
```

Question 2

In part (a) the majority of candidates answered correctly, as follows:

```
(i)   YES
(ii)  YES
(iii) NO
(iv)  YES
(v)   NO
```

```
(vi)  YES
(vii) YES
```

The majority answered part (b) correctly. However, a large minority failed to understand that the answer would be `L` since in integer division the result of 16 divided by 11 is 1, thus giving the character in position 1 in `ELI` (Strings are subscripted from 0).

In part (c) most candidates got their answers almost completely correct, as follows:

```
(i)   12 STARS
(ii)  5 STARS
(iii) infinite loop
(iv)  22 stars
(v)   infinite loop
(vi)  no output
(vii) for(int i=5;i>0;i--); System.out.println("**");
//semi-colon ends for loop, two stars printed.
```

One right answer seen.

```
(viii)      10 stars
```

In part (d) the most common mistake seen was that candidates would make the following statement (or similar) their first one: `int shortest=arr[0].length();`

This would mean that if the array was empty the program would have a run time error due to attempting to access an array element that did not exist. The most serious mistake seen was that some candidates made an array local to their method and then searched this empty array. This was a misunderstanding of the basics of programming using methods.

Many candidates wrote an answer similar to that given below, that was logically and syntactically correct.

```
static int shortestLength (String [] arr){
   if (arr.length == 0) return 0;
   int shortest=arr[0].length();
   for (int i=0;i<arr.length;i++){
      if (arr[i].length() < shortest) shortest=arr[i].
      length();
   }
   return shortest;
}
```

A minority of candidates forgot to return the answer, instead printing it to standard output.

Question 3

Part (a)(i) was almost always answered correctly, although a tiny minority wrote that the problem was that `int` variable *s* had not been initialised. The correct answer was given in various ways, such as `String` cannot be converted to `int`, or the statement `s = 1;` should be `s = "1";`. Provided that the explanation showed understanding that the String variable was having a number assigned to it, full credit was given.

In part (a)(ii) all candidates successfully identified the error that the variable *t* was being defined twice within the same scope. Some answers were a little unclear, with statements like 'the boolean t is the problem'.

In part (a)(iii) a minority thought that class *F* would not compile, because the *z* variable was static. While a majority correctly identified *D* as the class that would not compile, many of these candidates said that the problem was that the variable *z* was declared outside the main method. For full credit candidates needed to write that as *z* was an instance variable it could not be accessed directly by main, (i.e.it could not be accessed in a static context). A few candidates lost credit because they did not give a reason why *D* was their answer.

Part (b) was not very well answered on the whole. In part (b)(i) either:

```
if ((t1 > t2) && (b==true)) t1=t2+1;
```

OR

```
if ((t1 > t2) && (b)) t1=t2+1;
```

were marked as correct, when seen, which was not often. Some candidates included in their answer: `b = t1 > t2`; even though there was no evidence in the statements given that that the value of *b* depended on *t1* and *t2*. Many candidates wrote a `while` loop that would break once the desired behaviour had been achieved, `[if (t1 > t2) while (b) {t1=t2+1; break}]` which meant that the statements would behave the same, but adding a completely unnecessary loop is not good practice.

Part (b)(ii) was again answered mostly incorrectly. Candidates could, for full credit, have answered:

```
if (b==true) y = 17;
```

OR

```
if (b) y = 17;
```

The most common mistake was to include `y = y*10`; in the answer. Careful reading of the statements given would have shown that this expression could not be executed.The most common answer given was:

```
if (b==true) y = 7 + 10;
```

and another common answer that also gained some credit (although not much) was:

```
y = 7 + 10;
```

In part (b)(iii) most candidates could successfully rewrite the while loop as a for loop, however one common mistake is commented below:

```
public static int CoinToss1(int n){
   int sum = 0;
   //int i = 0; this statement should be deleted when the i
   variable is declared within the for loop as below
   for (int i = 0; i < n; i++){
      int r = (int)(Math.random()*2);
      sum = sum + r;
   }
   return sum;
}
```

In part (c) most candidates drew completely correct answers; one wrong answer seen more than once is noted below:

```
Right               Wrong
*                   *****
**                  ****
***                 ***
****                **
*****               *
*****               *
****                **
***                 ***
**                  ****
*                   *****
```

Question 4

In part (a)(i) some candidates thought that the output would be nothing, or the file name, but most understood that the output would be the text contained in *filey.java*.

In part (a)(ii) most understood that the variable *t* would get the value -1 when the *bling()* method reached the end of the file

In part (a)(iii) most understood that the program would stop with an uncaught `FileNotFoundException`, although a few said the program would stop with an error or an exception, or would crash. These somewhat vague answers received some credit.

All candidates knew that the answer to part (b) was ‘g’.

Most successfully answered the first part of (c) with:

```
s.charAt(1)
```

The expected answer for the part (c)(ii) was:

```
s.charAt(s.length() - 2)
```

The above answer was almost never given. Most candidates wrote an expression that would give the last character in a `String`, rather than the character immediately before the last.

In part (d) candidates usually gave answers similar to the following:

```
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]);
  }
}
```

One serious mistake was failing to write statements to open and read from the file. Other mistakes, most of them minor, includined:

- `FileReader` without `BufferedReader` (bad practice)
- sending the output to a file instead of writing to standard output
- characters were swapped, but nothing was output (to a file or standard output)
- swapped all 'c's for 'd's but not all the 'd's for 'c's, or *vice versa*
- did not take a file name from the command line, (i.e. the equivalent of `cat(args[0])`; in the above was missing).

The most serious mistake was poor logic(the program would swap characters and print the replacement character, but then would print the original character too).

One very original, correct, answer was:

```
import java.io.*;
public class swapcd{
   public static void readAndSwap(String s) throws Exception{
      BufferedReader in = new BufferedReader(new FileReader(s));
      int t = in.read();
      while(t !=-1){
         if(((char)t)=='c') t++;
         else if (((char)t)=='d') t--;
         System.out.print((char)t);
         t = in.read();
      }
   }
   public static void main(String[] args) throws Exception{
      readAndSwap(args[0]);
   }
}
```

Question 5

In part (a) candidates showed good understanding of static methods and variables. Most understood that static methods cannot operate on instance variables and that static methods can run before an instance of the class is made. Many appreciated that static variables hold the same value for every instance of the class and that instance variables **do not** hold the same value for every instance of the class. A few appreciated that static methods are often used for utility methods.

To answer (b)(i) candidates were given the following statements as a hint, not as an instruction, though most treated it as such:

```
Random r= new Random();
int randomA = r.nextInt(6);
```

The examiners had expected that at least a few candidates would understand that using the length of the answers array as a variable (see below) would be better than using 6 to get the random value.

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

One very common mistake was candidates not appreciating that as the *eightBall()* method was designed only to operate on the *answers* array, the array did not need to be a parameter of the method. Another common error was failing to return a `String` from the method, even though the question explicitly asked for this.

In part b(ii), all candidates could successfully write a main with a test statement, writing something similar to the answer below:

```
public static void main (String[] args){
   System.out.println(eightBall());
}
```

However, a test statement such as the one above would give a compilation error when the candidate had written an *eightBall()* method that was void and did not return a `String` (or anything else).

In part (c)(i) candidates were asked to write an instance method. This method was very similar to the *eightBall()* method, key differences being the name and the lack of the static keyword in the heading:

```
public String aToss(){
   Random r= new Random();
   int randomA = r.nextInt(tosses.length);
   return tosses[randomA];
}
```

Most candidates received no credit at all for (c), completely failing to understand the difference between static and instance methods. Candidates usually wrote a static method in answer to (c)(i). Even those candidates who answered the first part correctly went on to write a main method for part (c)(ii) that did not first make an instance of the *SixSidedDieToss* class in order to test the method. A correct answer to (c)(ii) would be as follows:

```
public static void main (String[] args){
   SixSidedDieToss toss = new SixSidedDieToss();
   System.out.println(toss.aToss());
}
```

In part (d) candidates either wrote good answers, as in the model answer below, or had no real idea how to approach the problem. Of those candidates who wrote good answers, the most common mistake was forgetting to make sure that two spaces were **not** inserted after the final character in the String.

```
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);
}
```

Question 6

In part (a) mostly correct answers were given, although for full credit candidates should have written each item of output on a new line, as follows:

```
0
1
2
3
4
```

A very small minority lost credit by missing out the 4 from the end, or adding a 5 to the output.

In (a)(ii) full credit was given to candidates who wrote that an `ArrayIndexOutOfBoundsException` would be thrown when the program was run; a few candidates lost some credit by stating that an error would happen, or an exception would be thrown. A few candidates incorrectly answered that the program would store 2 at position 5 in the array.

In part (a)(iii) almost all candidates understood that control would pass to the `catch` block and hence 'hello' would be the output.

In part (b) most candidates wrote a correct constructor, although a minority wrote a constructor with no arguments (no parameters) which would not have made the statement in the main method legal. A correct answer would be similar to the following:

```
public GolfClubMember(String first, String familyN, double
handicap, boolean annualFeesPaid){
   firstName = first;
   familyName = familyN;
   this.handicap = handicap;
   this.annualFeesPaid = annualFeesPaid;
}
```

In part (c) there were many different approaches to the logic of the program, although one popular approach was:

```
static void reverse(String word) {
   int length = word.length() - 1;
   String s = "";
   char c;
   for (int i = length; i >= 0; i--) {
      c = word.charAt(i)
      s += c;
   }
   System.out.println(s);
}
```

One common error was attempting to return the reversed `String`; the method heading included the 'void' keyword - hence the method should not be returning anything. Candidates who made errors of syntax were given most of the credit provided that the logic of their method would work. Logic errors lost candidates more credit, the most common were:

- the method made a local variable with the same name and type as the parameter. Often this parameter was not initialised (given a value) so the method was operating on an empty (null) `String` (logic error) although the method would not compile as local variables in methods must be initialised (syntax error)

- an out of bounds error would be thrown because the candidate failed to appreciate that the character at position `word.charAt(word.length())` did not exist since Strings, like arrays, are subscripted from zero
- the method would print one character only because the reversed char was not being concatenated (added to) the other reversed chars
- at each iteration of the loop the new `String` being made consisting of reversed chars was overwritten completely and not added to, so in the end only one character was returned
- the `for` loop was going in the wrong direction (`i--` should be `i++` or *vice versa*) meaning that reversing was not happening.