# Examiners' commentary 2017–2018

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

### General remarks

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

### Comments on specific questions

#### Question 1

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

(A) `while`

(B) `for`

(C) no loop needed

(D) nested `for` loops

Most candidates gained some credit, but almost all gave at least one of the four common mistakes, listed below:

- For (A) many answered 'no loop' since the user only has to enter one number. However, it would be more appropriate to use a while or a do/while loop in order to be able to reject invalid entries and ask the user to try again.
- For (B) some answered for loop, but a loop that is limited in the number of iterations it provides is not suitable for the open-ended task of soliciting correct entry from the user.
- For (C) most candidates understood that no loop was needed, as the final entry in an Array can be directly accessed, but a surprisingly large minority thought that a for loop was needed.
- For (D) only a minority understood that nested for loops would be the most appropriate solution.

(ii) The majority of candidates chose the correct answer from the given multiple-choice options, although a minority chose (B) 'The final expression'. The correct answer for part (ii), was:

(C) The guard

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

(B) Infinite loop – the standard output will fill up with "Hello world", one to a line, and this will continue indefinitely.

A few candidates thought that the program would not compile, and another

minority wrote that the program would compile, but since there was no final expression the loop would not run. Both these mistakes could have been avoided by reading the subject guide, section 8.4, which explains that `for(;;)` is syntactically correct.

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

(i) 9 asterisks

(ii) 6 asterisks

(iii) 6 asterisks

(iv) no output

(v) Infinite loop

(vi) 5 asterisks

(vii) 5 asterisks

(viii) Infinite loop

(ix) Infinite loop

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

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

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

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

Candidates were expected to read the *Verifier* class, and understand that the *getUserInput()* and the *isValidCustomer()* methods were missing. Almost all candidates understood that these were the two methods that they needed to write. Common mistakes were:

- missing parameters (both methods should have a single `String` parameter)
- wrong method type (methods were often `void`)
- no return statement (even some methods with the correct `String` or `boolean` type returned nothing).

Candidates should have understood, from the expression `if(isValidCustomer(customer))` in the *verify()* method, that the *isValidCustomer()* method takes a `String`, and will return `true` or `false`, hence the *isValidCustomer()* method should have a `String` parameter, be of type `boolean` and return a `boolean` value. All methods return a value

compatible with their return type, unless that type is `void` when they return nothing.

In the *getCustomerFromUser()* method there is a single statement `return getUserInput("Enter customer name: ");` Candidates were expected to understand from this that the *getUserInput()* method takes a `String`, since the statement is calling the method with a literal `String` parameter. Since the statement is a return statement, and since the *getCustomerFromUser()* method is of type `String`, candidates were also expected to understand that the *getUserInput()* method then must also return a `String`, that will then be returned by the *getCustomerFromUser()* method. Hence the *getUserInput()* method must take a `String` parameter, be of type `String` and return a `String`. Candidates were also expected to understand that the *getUserInput()* method should print its `String` parameter, and then attempt to read in user input, and that the *isValidCustomer()* method should search the customers `Array` for its `String` parameter, returning `true` if the parameter is in the `Array`, and `false` otherwise.

The most common logical errors were:

- The *getUserInput()* method did not print the message to the user given as its `String` parameter, or printed it after the `Scanner` object had attempted to read user input, i.e. the method first read user input, then asked for the input it had just read.
- The search logic for the *isValidCustomer()* method was often incorrect. Candidates would do one of two things:
    - With each iteration of the loop, if the search item is found return `true` else return `false`. This means that the first iteration of the loop ends the method, since it will always return either `true` or `false`, and since once a method returns something the method ends. Hence the search item can only be found if it is the first item in the array.
    - With each iteration of the loop a `boolean` variable is made `true` if the search item is found, and `false` if it is not found. Suppose the search item is found with the second iteration of the loop. This means that the `boolean` becomes `true`, but with the next iteration of the loop it becomes `false`. In fact the final loop iteration will determine the output of the variable returned. Hence the search item can only be found if it is in the last place in the array.

A very few candidates gave answers that were completely wrong; these candidates were clearly guessing and were very badly prepared for the exam.

## Question 2

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

| | | |
|---|---|---|
| (i) | (A) | Correct the first error only and recompile |
| (ii) | (A) | Public should be public |
| (iii) | (C) | `'else' without 'if'` |
| (iv) | (A) | `missing return statement` |

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

Part (i) a very small minority gave the correct answer; most candidates thought that the right answer would be to correct all errors and recompile. A few candidates even wrote that in practice they would correct the first error and recompile, but that the correct thing to do would be to correct all errors first. The subject guide, volume 1, section 2.8.1, states:

> *The best way to correct [compilation errors] is just to correct the first one and*

*then recompile. This is because the first error sometimes makes the compiler think there are lots of other errors which are not really there.*

For example a missing class closing bracket would give rise to many errors, as the compiler tries to interpret your code. Quite often adding the closing bracket will make all these errors disappear.

For part (ii) a surprisingly large minority did not understand that 'Public' should be 'public'. Many thought that 'Args' should be 'args', clearly not understanding that 'Args' is the name given to the `String Array` that would hold any command line input. As such it is a variable name, normally called 'args' by convention, but the compiler does not care what you call it. Others thought that there were statements missing from the main method, which there are, but again the compiler does not care about this. A row of semi-colons just denote many empty statements as far as the compiler is concerned.

In part (iii) many candidates did not understand that without brackets enclosing the two separate statements: `x=1;` and `y=2;` the if statement ended with `x=1;` Therefore, as far as the compiler was concerned, the else statement was not part of the if statement, giving rise to the compilation error:

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

In part (iv) it was disappointing to see that some candidates did not appreciate that a typed method must return a variable of its type. Therefore the `add()` `method, being of type int, should have returned an int`.

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

(i) NO

(ii) NO

(iii) YES

(iv) YES

(v) YES

(vi) YES

(vii) YES

(viii) YES

(ix) YES

Many candidates answered part (b) entirely correctly, with no candidate achieving less than five out of nine. There were no common errors.

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

```
public static String sentenceCase(String s){
   if (s == null) return s;
   if (s.length() < 2) return s.toUpperCase();
   return s.substring(0,1).toUpperCase() + s.substring(1).
   toLowerCase();
      }
```

***or***

```
public static String sentenceCase(String s){
      if(s==null||s.length()==0) return "";
      String a = s.substring(0,1).toUpperCase();
      a+=s.substring(1).toLowerCase();
      return a;
}
```

***or***

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

The first two answers above deal with `Strings` that may be empty, or `null`. Very few candidates dealt with the empty or `null String` in their answers, and those that did are congratulated for their excellent answers. Excluding the empty `String` avoids a run-time error (`StringIndexOutOfBoundsException`) that would otherwise be triggered by the calls to the `String.substring()` methods in the second return statement. Excluding the null String avoids throwing a `NullPointerException`. However, no credit was given for explicitly dealing with the cases of empty or `null Strings`, hence answers such as the third answer above received full marks.

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

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

Since `String.charAt()` returns a `char` value, this gives the compilation error:

```
error: incompatible types: char cannot be converted to String
```

Another error was attempting to use a String method, such as *toUpperCase()*, with a `char`.

```
char c = (s.charAt(0)).toUpperCase();
```

Giving the compilation error:

```
error: char cannot be dereferenced
```

Others tried casting, for example:

```
String v=((String)s.charAt(0));
```

Resulting in the error:

```
error: incompatible types: char cannot be converted to String
```

Another error, although the examiners did not deduct marks for it, was attempting to exclude numbers as input by seeing if the input `String` could be parsed to an `int`, using `Integer.parseInt()`, and not attempting to change the case of the `String` if it could. This was perhaps badly thought

out, as a `String` may be any combination of letters and numbers, for example "hello32", in which case parsing to an `int` will fail. In addition, logic to exclude values that could be parsed to an `int` was not necessary, as the *toUpperCase()* and *toLowerCase()* methods will simply return any `String` that includes numbers or other non-alphabetic characters by changing the case of any letters found, and ignoring other characters. If nothing is found to convert, the methods will just return the `String` without changes.

A minority of candidates wrote ingenious methods to determine if a `String` was a combination of letters and numbers, and find what part of the `String` was a letter, and which parts were non-alphabetic characters. It was a pity to see candidates wasting their time in this way, since no credit could be given.

Other errors seen were numerical mistakes with the `String.substring()` methods, such as `substring(1,2)` for the first character, which, since `String` characters are numbered from zero, will give the second character, not the first. Another mistake, was instead of using `substring(1)` to return the `String` without its first `char`, some used `substring(1, s.length()-1)`, which would return the String without its final char, and others `substring(1, s.length()+1)`, which would give a runtime error (`StringIndexOutOfBoundsException`). It was surprising that some candidates did not understand how Java subscripts `Strings`. Note that some candidates used `substring(1, s.length())`, which is correct.

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

## Question 3

**a.** (i) The programme will not compile because `x > y` should be contained in brackets, i.e. `(x > y)`.Most candidates answered (i) correctly, while a minority thought that the problem was that x and y were uninitialised, showing that they did not understand that class variables such as x and y are given default values. In the case of an `int`, the default is 0, so the logic becomes if 0 is greater than 0, print "error". A few thought that the main method would not be able to access x and y, which would be true if x and y were instance variables, main would not be able to directly access them. Since they are static (i.e. class) variables, main can directly access both variables.

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

(A) `hello`

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

(B) `false`

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

(A) The program will output `hello`

Again, this part was answered correctly by most candidates, however a small minority were confused, and seemed to think that the expression `(t==0||t==3)` meant "if t is zero AND t is 3", rather than what it actually means, which is "if t is zero OR t is 3". These candidates thought either that since it was impossible for t to be 0 and 3 the program would output nothing, or that the program would not compile. Thinking that the program would have no output if the expression was ANDing, was correct. Thinking that the program would not compile if the expression was ANDing was a more serious error. The compiler does not have the capacity to flag as an error

that the `true` condition is impossible to reach as an `int` variable cannot simultaneously hold two different values.

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

(i) 1+1

2

11

(ii) 1

1.5

1.5

(iii) /*1*/ and /*3*/

In part (i) almost all candidates gave the correct answer, showing that they understood the difference between integer addition and `String` concatenation. In part (ii) again most answers were correct, showing that candidates understood the difference between integer division, and division with real numbers. Candidates also understood that the Java compiler will automatically choose the correct division based on the type of variables it is given. Part (iii) was answered correctly by almost every candidate.

Only one common error was seen in part (ii), a few candidates thought that the result of the third division would be 1, rather than the correct value of 1.5.

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

```
import java.util.Scanner;

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

   public static void main(String[] args){
      question();
   }
   public static void question() {
      Scanner scanner = new Scanner(System.in);
      String input = "";
      System.out.println("Human beings glow in the dark.");
      System.out.println(s);
      input = scanner.nextLine();
      boolean answer1 = Boolean.parseBoolean(input);
      System.out.println("A piece of paper can only be
         folded in half 8 times.");
      System.out.println(s);
      input = scanner.nextLine();
      boolean answer2 = Boolean.parseBoolean(input);

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

This question was intended to test candidates' grasp of Boolean logic. Very few candidates gained full marks for this question. The most common logical error was firstly to evaluate if the first answer and the second answer are both true, and output "You are so right" if they are. This would then be followed with a statement to evaluate if the first answer or the second answer was true, and output "You are half right" if either one was. This would look like the following:

```
if(ans1 && ans2) System.out.println("You are so right!");
if(ans1 || ans2) System.out.println("You are half right");
```

The problem with this logic is that the second `if` condition will evaluate to `true` when one or both answers are true. This means that where both answers are true the statement `if ((answer1 || answer2)` will evaluate to `true` and so the user would see both the statement `You are so right!` and the statement `"You are half right"`. Hence the second statement needs to be 'else if' in order to exclude any cases covered by the first statement.

Other errors seen were syntactic errors in the use of the `Boolean.parseBoolean()` method, and not using `Boolean.parseBoolean()` or equivalent logic to determine true and false values from the user's input. In particular, directly comparing the `String` entered by the user to "true" was treated as an error. The user may have entered "true", or "True" or "TRUE". The `Boolean.parseBoolean()` method would return `true` with any of these inputs, reflecting the user's clear intention, but a statement such as `if answer1.equals("true")` would only return `true` for one of those inputs.

Some good answers were seen; the best answers broke the class down into methods.

## Question 4

**a.** The correct answers from the available multiple-choice options to parts (i-iv) are as follows:

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

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

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

(iv) (B) FileNotFoundException

In parts (i) and (ii) and (iv) 99% of answers seen were correct. In part (ii) some candidates correctly noted that the answer (C) was dependent on the file path being correct. In part (iii) most candidates gave the correct answer, while a few thought that the answer was (A) 'The program would hang, waiting for input', (B) or 'None of the above'.

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

(i) `gamma`

(ii) `Proceed`

(iii) (D) None of the above

(iv) `NumberFormatException`

Parts (i) and (ii) were answered correctly by most candidates. In part (iii) a minority answered (C) thinking that the program would output the numbers 0 to 6, one to a line. These candidates should have understood that (C) could not be the right answer for two reasons. One reason is that (C) gives 7 lines out output, while the `Array` only contains 6 items. The other is that the `Array`'s first value will be 1, so 0 cannot be output. Another minority thought that the program would, when run, throw an `ArrayIndexOutOfBoundsException`. In fact the output would be:

1

2

3

4

5

6

Part (iv) was the answered wrongly about half the time, with candidates giving such answers as:

- TextFormatException
- Reached end of program while parsing
- InvalidArgumentException
- Throws Exception

However, by far the most popular wrong answer was that a `String` cannot be parsed to an `int`. These candidates had clearly not read the subject guide, volume 1, section 9.6 Parsing Strings that Represent Integers, or section 9.6.1 `Integer.parseInt()`

**(c)** A model answer for this would be as follows:

```
public static void divideFileContents(){
   int x;
   try {
      Scanner in = new Scanner(new
            FileReader("num1.txt"));
      PrintStream out =new PrintStream(new
            FileOutputStream("num2.txt"));
      while(in.hasNextLine()){
            String line = in.nextLine();
            x = Integer.parseInt(line);
            x = x / 10;
            out.print(x+"\n");
      }
      in.close();
      out.close();
   }catch(Exception e) {
      System.err.println("Error: Could not open or save");
   }
}
```

A very small minority of candidates, less than 10%, gained full marks for this question, which was perhaps the most badly attempted question on the paper. Numerous mistakes were seen, but all were of two types: (1) errors with the streams for reading and writing to the files and (2) errors with the try and catch blocks.

The question asks that numbers are read in from the *num1.txt* file, meaning that the numbers in the file need to be read in as Strings, and then parsed to `int`s. However, many candidates lost marks by copying the statements in the *bling()* method that read in from the file by using the `BufferedReader.read()` method to read in `char`s. What was needed was to read in each line as a whole; hence, the model answer above uses the `Scanner.nextLine()` method, to read in a line at a time, before attempting to parse the line to an `int`, divide it by 10 and save the result in another text file. Candidates can find information about reading in from a file line by line in section 7.6.3 of the second volume of the subject guide.

Many mistakes were also seen in streams to write to the file, candidates can consult section 7.6 of volume 2 of the subject guide for information about writing to a file, and why we should always close the file after writing to it (to be sure that our data has been saved).

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

## Question 5

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

(i) There is a simple rule to distinguish primitive from reference variables in Java – all primitives start with a lower case letter. Applying this rule, the answer to (i) was char and int, given by surprisingly few candidates. So many varied and incorrect answers were seen that it seems that most candidates were guessing, and had not read section 4.4 of volume 1 of the subject guide which lists the 8 primitive types, or section 5.4 of volume 2 of the subject guide that discusses simple (primitive) and reference variables.

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

(A) TRUE

(B) FALSE

Again, only a minority of candidates gave the correct answer, with most candidates thinking that (A) was false. Some also thought that (B) was true, when it is false because Java allows 16-bits to store `char`s, see section 6.5 of volume 2 of the subject guide.

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

```
hello
```

The majority of candidates answered this part correctly, showing that candidates understood that giving a `String` variable as a method parameter did not change the value pointed at by the `String` variable.

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

7

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

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

```
(char)c
```

Candidates found part (i) puzzling, and most clearly did not understand that in the statement `System.out.println("You typed in "+(char)c);` an `int` variable was being cast to a `char` with `(char)c`. Some candidates made no attempt, others gave answers that were clearly guesses such as "throws IOException". Casting is covered in volume 2 of the subject guide, section 6.6.

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

(B) 107 is the Unicode value of the first `char` the user entered

There was more confusion in part (ii), where most candidates answered that the user had entered 107, although quite a few chose (D) None of the above. In fact, `System.in.read()` reads a single character, as the subject guide, volume 2, discusses in section 6.7. Since `System.in.read()` reads a single character at a time, it is not possible that the user entered 107. Instead the correct answer is that 107 is the Unicode value of the first char the user entered. A (rare) correct answer demonstrated that the candidate understood that `System.in.read()` reads in one `char` at a time, and that a `char` is both an `int` and a keyboard character.

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

```
public class Unicode{
   public static void main(String [ ] args){
      System.out.println((char)37);
   }
   }
```

Despite the confused answers to parts (i) and (ii), part (iii) was normally answered correctly. Candidates lost marks here for simple things such as just writing a main method or a single statement when a class was asked for, despite having written a correct statement to display the `char` with the Unicode value 37.

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

(B) 121

y

The great majority of candidates could not give a correct answer to part (iv), where the 'most likely output of the *Chars1* class' was asked for. The most popular answer was (C) *No output – compilation error*. (B) is the correct answer because in the statement `System.out.println('8'+'A');` the single quotes mean that the variables are `chars`. Since a `char` can be an `int` or a character from the keyboard, the JVM treats them as `ints` and adds them. The Unicode value of '8' is 56, and that of 'A' is 65, giving 121. Of course, candidates were not expected to know that the answer would be 121, but to understand that the first line of output would be an integer.

In the second line the statement `System.out.println((char)('8'+'A'));` is casting the `int` value 121 to a `char`. Again 121 comes from the compiler treating '8' and 'A' as `ints`, adding them and getting 121. The result of the cast is 'y'. Again, candidates were not expected to know that 'y' would be the output, but that a `char` would be the output. Since (B) was the only answer with an `int` followed by a `char`, it was the most likely output of the Chars1 class.

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

```
import java.util.Scanner;

public class Q5Answer {

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

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

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

Some correct answers were seen, most of which varied in small ways since various correct answers were possible. Most correct answers divided the work of the class among more than one method.

## Question 6

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

(A) Constructors are called by their containing class name by convention **FALSE**

(B) Constructors have return types **FALSE**

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

(D) A class can have arbitrarily many constructors **TRUE**

(E) The keyword `this` can be used in a constructor to distinguish the formal parameter from the field name **TRUE**

(F) An instance method does not have the keyword static in its heading **TRUE**

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

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

The majority of candidates gave most answers correctly, with the exception of (A), which was answered wrongly by nearly all candidates. Two candidates correctly made the point that (G) was only true when the fields of the parent class were ***not*** private, which the question should perhaps have incorporated.

**b.** Some candidates gave the fragment numbers in the correct order as their answer, which, for full marks, could be:

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

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

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

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

```
4. public class ToDoList {
      private String item;
      private String importance;
9.    private boolean done;
8.    public ToDoList(String item,String i,boolean done){
10.         this.item = item;
            this.importance=i;
            this.done = done;
      }
6.    public boolean isDone() {
7.          return done;
      }
2.    public String toString(){
            String s = ("Action: "+item+" with ");
            s = s+(importance+" importance");
            s = s+("\nAction taken status: "+done);
5.          return s;
      }
```

```
3.    public static void main(String[] args) {
            //test statements
1.          ToDoList toDoList = new ToDoList("Submit 109
                  cwk","high",true);
            System.out.println(toDoList);
      }

11. }
```

It was pleasing to see that most candidates correctly answered this question. There were a few errors, the only common ones being putting the statement `this.done = done;` in the *isDone()* method rather than in the constructor, and placing the class closing bracket (fragment 11) somewhere in the body of the class, rather than right at the end.

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

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

Some mistakes were seen, including: (1) some candidates added the method logic for *isContained()* directly to the *Exists* class, rather than extending *Exists* to a new class; and (2) others did extend *Exists* to a new class, *StringExists*, but instead of writing an *isContained()* method they wrote the logic that should have been in the *isContained()* method in a main method.

Some good answers were seen, with candidates correctly extending *Exists* and implementing *isContained()*, many of these candidates also wrote constructors for the class, which was very good, although not necessary for full credit.