Please fill in your Student Number and, optionally, Name.	
Student Number	**:** ____________________
Name	**:** ____________________

For Official Use	
Mark	**:** __________
Marker	**:** __________

University of Cape Town ~ Department of Computer Science

Computer Science 1015F ~ 2007

Supplementary Theory Test 2A Solution

Marks : 30

Time : 40 minutes

Instructions:

a) Answer all questions.

b) Write your answers in the space provided.

c) Show all calculations where applicable.

Question 1: Multiple Choice. [4]

For each question, write down ONLY the letter of the correct answer.

a) The Java branching mechanisms are: [1]

A. `if-else` statements

B. `switch` statements

C. `while` statements

D. A and B

Answer: ________________________________

D

b) A mystery Java operator, called '✰', has the following truth table: [1]

A	B	A✰B
true	true	true
true	false	true
false	true	true
false	false	false

This operator is actually:

A. !

B. ||

C. &&

D. None of the above

Answer: ________________________________

B

c) Which of the following operators has the *highest precendence*? [1]

A. !

B. ||

C. &&

D. ==

Answer: ________________________________

A

d) Examine the following Java code: [1]

```
boolean A=false, B=false, C=false, D=true;
System.out.print( ! D || C);
System.out.print(A || B == C && D);
```

When executing this code, the output is:

A. truetrue

B. truefalse

C. falsetrue

D. falsefalse

Answer: ___________________________________

C

Question 2: Short questions [4]

a) What is *short-circuit evaluation* and why is it useful?

[2]

*Java can take a shortcut when the evaluation of the first part of a Boolean expression produces a result that evaluation of the second part cannot change. e.g. when evaluating two Boolean subexpressions joined by **&&**, if the first subexpression evaluates to **false**, then the entire expression will evaluate to **false.** For ||, if the first part evaluates to true, then the whole expression is true.*

There are times when using short-circuit evaluation can prevent a runtime error, and it is faster.

a) Write down the exact output of the following code.

```
  public static void main(String[] args)
  {
   int a = 3;
   switch (a)
   {
   case 2:
      System.out.println( "Me" );
       break;
   case 3:
   case 1:
      System.out.println( "Do" );
   default:
      System.out.println( "Re" );              }
    }
}
```

[2]

Do

Re

[only 1 mark if a minor error, like putting htem on the same line, or if just "Do"]

Question 3:Longer questions[12]

a) Examine the main method listed below:

```
public static void main(String[] args)
  {
   for(int i=1;i<=100;i++)
   {
     int res = i*5;
     if((res%4>0)&&(res%3>0)) continue;
        System.out.println(i + " times 5 = " + i*5);
   }
  }
```

i. Describe what this method does – i.e. the output that it produces

[2]

*It calculates the 5 time table, up to the first 100 terms. However, only those values that are divisible by **both** 4 and 3 are printed.*

ii. Convert the method above to use a *for* loop instead of a *while* loop. **[4]**

```
int i=0;
while(i<100)
{
 i++;

 int res = i*5;

  if((res%4>0)&&(res%3>0)) continue;
  System.out.println(i + " times 5 = " + i*5);
}
```

a) Now write a program to draw a square frame of a certain height, supplied by the user.

e.g. If the user supplied a height of 1, the output will be:

```
*
```

If the user supplied a height of 3, the output will be:

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

If the user supplied a height of 5, the output will be:

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

And so on. You are given the outline of the program, just supply the missing lines of code**.**

```
import java.util.Scanner;
   public class mystery
   {
      public static void main(String[] args)
      {
        Scanner keyboard = new Scanner(System.in);
        System.out.println("Enter the height of the frame:");


       int height = keyboard.nextInt();
        for(int stars=1; stars<=height; stars++)
           System.out.print('*');  [1]
        System.out.println();      [1]
        for(int row=2; row<height; row++)
        {
         System.out.print('*');      [1]

          for(int spaces=1; spaces<height-1; spaces++)
            System.out.print(' ');    [1]

         System.out.println('*');     [1]
         }
         for(int stars=1; stars<=height; stars++)
           System.out.print('*');      [1]



      }
   }
```

[6]

Question 4: Testing [5]

a) In the context of testing, explain what an equivalence class is. [1]

It is a collection of values for which it is expected that the program behaves in the same way [1].

b) Suppose you are testing the following code. Indicate a set of test values that may be used if you are using enforcing statement coverage. Then indicate what additional test values may be used to check path coverage. [4]

```
if (x < 100)
  y = 1;
else
  y = 2;
if (a < 200)
  b = 1;
else
  b = 2;
```

Statement coverage:

x = 10, a = 10 [1]

x = 200, a = 300 [1]

or any pair of test cases to run each statement at least once.

Path coverage:

x = 10, a = 300 [1]

x= 200, a = 10 [1]

or any pair of two test cases for the other two possibilities

Question 5: Object Oriented Programming [5]

a) What is the difference between an object and an instance? [1]

No difference! [1]

b) Write a statement to invoke the **subtract** method on an instance named **calculator**, with a single integer parameter with value **17**. [2]

calculator.subtract (17)

Dot notation [1], parameter [1]

c) How do instance variables differ from local variables? [2]

Instance variables are persistent across all methods for an object. Local variables are only available while a method is being invoked [2].

Or

Instance variables are associated with or declared in classes/objects while local variables are associated with or declared in methods. [2]