Question 1

a) In this question you are asked to write a Java class to represent the concept of probability. This class must hold a single variable representing the probability of something occurring, e.g. a probability of 0.1 indicates that there is 10% chance of something happening.

You must write the entire **Probability** class, including: TWO constructors (**Probability**), get/set methods (i.e. **setProbability()** and **getProbability()**), and the **toString()** method. The skeleton structure of the class is given in **Figure 1** – several parts of the code are missing, which you must fill in. These are indicated by the gaps (labelled **A–I**) and **TODO**.

Important:

- You must write the access modifiers and return types, for all the methods/constructors.
- You must also write the body of the methods (**TODO** means that you need to write something in the method body).
- When setting the probability value, your class code must ensure that the value set is between **0** and **1** inclusive (i.e. it must be **p>=0** and **p<=1**, where **p** is the probability).

[15 marks]

```
public class Probability {
   ____A____ ____B____ probability = ____C____; // variable to store
                                                 // probability
                                                              (1.5 marks)
   public Probability(____D____) { TODO } // constructor taking
                                           // numerical value
                                                              (2 marks)
   public Probability(____E____) { TODO } // copy constructor taking
                                           // Probability object
                                                              (2 marks)
   public ____F____ setProbability(____G____) { TODO } // set method
                                                              (4.5 marks)
   public ____H____ getProbability() { TODO } // get method
                                                              (2 marks)
   public ____I____ toString() { TODO } // toString method
                                                              (3 marks)
}
```

Figure 1

	Do not write in this column
`____A____ ____B____ probability = ____C____; // variable to store ...`	**1.5 marks**
A: `private` [0.5 mark]	
B: `double` [0.5 mark] (Note: A **float** (but not **int**) data type is also accepted.)	
C: `0.0` or `1.0` [0.5 mark] (Note: Other sensible default values are accepted.)	

	Do not write in this column
`public Probability(___D___) { TODO } // constructor taking ...`	**2 marks**
`D:` `double p` [1 mark]	
`TODO:` `this.setProbability(p);` [1 mark]	
(Note: **`probability = p;`** is also accepted.)	
`public Probability(___E___) { TODO } // copy constructor ...`	**2 marks**
`E:` `Probability p` [1 mark]	
`TODO:` `this.setProbability(p.getProbability();` [1 mark]	
(Note: **`probability = p.getProbability();`** is also accepted.)	
`public ___F___ setProbability(___G___) { TODO } // set method`	**4.5 marks**
`F:` `void` [0.5 mark]	
`G:` `double p` [1 mark]	
`TODO:`	
`if (p < 0 || p > 1) {` [1 mark]	
`  System.out.println("ERROR: Probability must be >=0 and`	
`<=1.");` [1 mark: sensible error message]	
`}`	
`else {`	
`  this.probability = p;` [1 mark]	
`}`	
`public ___H___ getProbability() { TODO } // get method`	**2 marks**
`H:` `double` [0.5 mark]	
`TODO:` `return this.probability;` [1.5 marks]	
`public ___I___ toString() { TODO } // toString method`	**3 marks**
`I:` `String` [1 mark]	
`TODO:` `return "" + this.probability;` [2 marks]	
	15 marks

b) Given the **`Probability`** class in *part a)* of the question, you <u>must</u> now write a series of *test code* to demonstrate that the class functions correctly. Write the following test cases:

[4 marks]

i) Write test cases to test that the constructors are correct. This <u>must</u> include the Java code, as well as a short description of what the code does.

(2 marks)

ii) Write test cases to test that the *set* method is correct. This <u>must</u> include writing the Java code, as well as a short description of what the code does.

(2 marks)

	Do not write in this column
i)	**2 marks**
CODE FOR TEST CASES:	
`Probability p1 = new Probability(-1);  // ERROR: Probability`	
`                                       // must be >=0 and <=1`	
`Probability p2 = new Probability(0.3); // NO ERROR`	
[2*0.5 mark: TWO sensible test cases]	
DESCRIPTION:	
The constructors should be tested with values outside the range **`[0,1]`** (which will	
produce a corresponding error message) <u>and</u> inside the range (which will function	
normally). [1 mark]	
ii)	**2 marks**
CODE FOR TEST CASES:	
`Probability p2 = new Probability(1);` [0.5 mark]	
`p2.setProbability(-0.01); // expect ERROR: Probability must`	
`                          // be >=0 and <=1`	
`p2.setProbability(0.3);   // NO ERROR`	
[2*0.5 mark: TWO sensible test cases]	
DESCRIPTION:	
The *set* method should be tested with values outside the range **`[0,1]`** (which will	
produce a corresponding error message) <u>and</u> inside the range (which will function	
normally). [0.5 mark]	
	4 marks

c) Briefly describe the concept of *encapsulation*, why it is important, and how it is implemented in a Java class. Is there any case where you would not encapsulate data? If so, explain your answer.

[3 marks]

	Do not write in this column
Encapsulation “hides” your data, protecting it and allowing it to only be changed by	
set methods, meaning that you have control over how it can changed. [1 mark]	
Encapsulation is achieved by marking variables as **`private`**. [1 mark]	
If you have a constant value (indicated by the keyword **`final`**), then it is not	
necessary to encapsulate it as that value will not change. [1 mark]	
	3 marks

d) In Java, there is a keyword **`abstract`**. Describe what the **`abstract`** keyword means, by referring to the TWO different ways in which it can be used.

[3 marks]

	Do not write in this column
If a class is marked as **`abstract`**, it means that it cannot be instantiated with the	
keyword **`new`** [1 mark], and must be extended to make a concrete class [1 mark].	
If method is marked as **`abstract`**, then it must be implemented in an inheriting	
class. [1 mark]	
	3 marks

Question marking: $\frac{\quad}{15}+\frac{\quad}{4}+\frac{\quad}{3}+\frac{\quad}{3}=\frac{\quad}{25}$

Question 2

a) The questions below refer to the concepts of inheritance and arrays in Java:

[9 marks]

i) What is *method overriding*? How do you override a method in a superclass?

(2 marks)

ii) Write a class *declaration* for a class called **`Lion`** that inherits from (i.e. extends) a class called **`Animal`**.

(1 mark)

iii) Write the Java constructor for the **Animal** class mentioned in *part ii)*. It should accept the name of the **`Animal`** as a parameter. You can assume that a method called **`setName(String name)`** already exists in the class **`Animal`**.

(2 marks)

iv) Write the Java constructor for the **`Lion`** class that extends **`Animal`** (first mentioned in *part ii)*). The constructor should also accept the name of the **`Lion`** as a parameter.

(1 mark)

v) Write a Java class called **`Zoo`**. It must contain an array that holds a list of **`Animal`** objects (first mentioned in *part ii)*) and a method called **`addAnimal`** that adds new **`Animal`** objects into the array.

(3 marks)

	Do not write in this column
i)	**2 marks**
Method overriding is the practice of re-defining the implementation of a method	
defined in the superclass [1 mark]. The method signature must be exactly the same as	
that of the superclass [1 mark].	
ii)	**1 mark**
`class Lion extends Animal` [1 mark]	
iii)	**2 marks**
`public Animal(String name) {` [1 mark]	
`setName(name);` [1 mark]	
(Note: Award only 0.5 mark if student did not use method **`setName()`**.)	
`}`	

	Do not write in this column
iv)	**1 mark**
`public Lion(String name) { super(name); }` [1 mark]	
v)	**3 marks**
`public class` **`Zoo`** `{`	
`  Animal[] animals;` [0.5 mark: array declaration]	
`  int counter = 0;`	
`  public` **`Zoo()`** `{`	
`    animals = new Animal[10];` [0.5 mark: array instantiation]	
`  }` (Note: It is also acceptable to declare and instantiate on the same line of code.)	
`  public void` **`addAnimal(Animal a)`** `{`	
`    Animals[counter] = a;` [1 mark]	
`    a++;` [1 mark] (Note: Any answer that doesn't override the previous	
`Animal` objects in the array, is acceptable.)	
`  }`	
`}`	
	9 marks

b) The questions below relate to interfaces in Java:

[7 marks]

i) Can an interface in Java contain a method *implementation*?

(1 mark)

ii) Imagine that you are writing the software for a graphics application (e.g. Microsoft Paint). Write the code for an *interface* called **`IGraphicsShape.`** This interface must include one **`void`** method called **`drawShape()`**.

(2 marks)

iii) Write a class called **`Square`**. This class must implement the **`IGraphicsShape`** interface written in *part ii)*. The **`drawShape()`** method should print out the message **`"Drawing Square"`** when invoked.

(2 marks)

iv) State TWO similarities between *interfaces* and *abstract* classes.

(2 marks)

	Do not write in this column
i)	**1 mark**
No [1 mark]	
ii)	**2 marks**
`public interface IGraphicsShape {` [1 mark]	
`   public void drawShape();` [1 mark]	
`}`	
iii)	**2 marks**
`public class Square implements IGraphicsShape {` [1 mark]	
`   public void drawShape() {`	
`      System.out.println("Drawing Square");` [1 mark]	
`   }`	
`}`	
iv)	**2 marks**
Both *abstract* classes and *interfaces*, [2 marks: any TWO of the list below]	
• cannot be instantiated;	
• can contain abstract methods;	
• are supertypes to classes that extend/implement them.	
	7 marks

c) The questions below relate to memory management in Java:

[9 marks]

i) Java maintains TWO memory spaces, the *Stack* and the *Heap*. Which memory space is used to store objects?

(1 mark)

ii) Objects in Java are automatically removed from memory by the *garbage collector*. When is an object marked for garbage collection?

(1 mark)

iii) Look at the code fragment in **Figure 2**. By the end of the code, is the **`String`** containing the word **`“Hello”`** eligible for *garbage collection*?

(1 mark)

```
String myString;
myString = new String(“Hello”);
myString = new String(“World”);
```

Figure 2

iv) Indicate TWO ways in which you can make the object containing the word **`“World”`** (in **Figure 2**) eligible for *garbage collection*.

(2 marks)

v) Look at the code fragment in **Figure 3**. What will be the output of this code? Explain your answer.

(3 marks)

```
String s1 = new String(“Hello”);
String s2 = new String(“Hello”);
if (s1 == s2)
   System.out.println(“Match!”);
else
   System.out.println(“No match!”);
```

Figure 3

vi) Explain the purpose of the **`equals()`** method in the **`String`** class.

(1 mark)

	Do not write in this column
i)	**1 mark**
It is the Heap. [1 mark]	
ii)	**1 mark**
When there is no reference to it. [1 mark]	

	Do not write in this column
iii)	**1 mark**
Yes – **“Hello”** has no reference to it. [1 mark]	
iv)	**2 marks**
Approach **1**: Set the object to **null**. [1 mark]	
Approach **2**: Re-assign the object to a different one. [1 mark]	
v)	**3 marks**
The output will be **“No match!”**. [1 mark]	
This is because the operator **==** compares the memory references [1 mark], instead of	
the **String** contents [1 mark].	
vi)	**1 mark**
It compares the content of the **String**s, rather than the memory references. [1 mark]	
	9 marks

Question marking: $\frac{}{9}+\frac{}{7}+\frac{}{9}=\frac{}{25}$

Question 3

a) The questions below refer to the concept of scope and lifetime of variables in Java:

[9 marks]

i) What is the scope of a local variable? Support your answer with a short coding example demonstrating the scope of a local variable.

(3 marks)

ii) What are the main differences between *local* variables and *instance* variables?

(3 marks)

iii) How do you refer to an instance variable when a local variable with the same name exists within the same scope? Provide a short coding example.

(3 marks)

	Do not write in this column
i)	**3 marks**
A local variable has block scope; that means within the brackets **{ }** only. [1 mark]	
CODING EXAMPLE: [1 mark]	
`if (a == 2) {`	
`   int b = 5;`	
`}`	
Variable **b** cannot be accessed outside of the **`if`** statement [1 mark].	
ii)	**3 marks**
Local variables have a block scope, whereas instance variables have an object scope	
(depending on the access modifiers used). [1.5 marks]	
Local variables need to be initialised with a value before used, whereas instance	
variables will be assigned with a value automatically if a value is not provided.	
[1.5 marks]	
iii)	**3 marks**
You can always refer to the instance variable by using the **`this`** keyword.	
[1.5 marks]	
CODING EXAMPLE: [1.5 marks]	
`String name;`	
`public void setName(String name) { this.name = name; }`	
	9 marks

b) The code fragment in **Figure 4** is an example of a class with a method that uses *recursion*:

[10 marks]

```
public class TestRecursion {
  public static long foo(long n) {
    if (n <= 1) { return n; }
    else { return n * foo(n – 1); }
  }
  public static void main(String args[]) {
    System.out.println(foo(3));
  }
}
```

Figure 4

i) Explain the concept of *recursion* in Java.

(2 marks)

ii) What is the output of the code in **Figure 4**?

(2 marks)

iii) Rewrite the **`foo()`** method so that it achieves the same output but uses *iteration* (i.e. a loop), instead of recursion.

(6 marks)

	Do not write in this column
i)	**2 marks**
Recursion is a technique in which a method calls itself a number of times, to solve a	
problem. [2 marks]	
ii)	**2 marks**
The output is **6**. [2 marks]	
iii)	**6 marks**
`public static long foo(long n) {`	
`  int result = 1;` [1 mark]	
`  for (int i=1; i<=n; i++) {` [2 marks]	
`    result = result * i;` [3 marks]	
`  }`	
`  return result;`	
`}`	
	10 marks

c) The questions below refer to wrapper classes in Java, which are used to represent primitive data types as objects.

[6 marks]

i) What *abstract* class do all numerical wrapper classes in Java extend?

(1 mark)

ii) Name TWO wrapper classes. Write the necessary code to instantiate each of them.

(3 marks)

iii) Provide an example of a case when a wrapper class is necessary.

(2 marks)

	Do not write in this column
i)	**1 mark**
All numerical wrapper classes extend the *abstract* class **`Number`**. [1 mark]	
ii)	**3 marks**
TWO examples of wrapper classes: **`Byte`**, **`Integer`**, **`Double`**, **`Short`**, **`Float`**	
[2*1 mark: TWO correctly identified wrapper classes]	
Code to instantiate a wrapper class:	
`Integer i = new Integer(10);` [1 mark: Any wrapper class is acceptable.]	
iii)	**2 marks**
When adding a primitive data type (e.g. **`int`**) in an **`ArrayList`**. [2 marks]	
(Note: Other sensible answers are accepted.)	
	6 marks

Question marking: $\frac{}{9} + \frac{}{10} + \frac{}{6} = \frac{}{25}$

Question 4

a) The questions below refer to exceptions and assertions in Java:

[7 marks]

i) What is the output of the program in **Figure 5** (lines numbered **1-13**) when *assertions* are disabled and when *assertions* are enabled? Justify your answers.

(4 marks)

```
 1   public class AssertionTest {
 2     public static void main(String[] args) {
 3       try {
 4         assert(false);
 5       }
 6       catch (Exception ex) {
 7         System.out.println("Caught an exception!");
 8       }
 9       catch (Error err) {
10         System.out.println("Caught an error!");
11       }
12     }
13   }
```

Figure 5

ii) The code in **Figure 6** (lines numbered **1-14**) is a modified version of the program in **Figure 5**. What is now the output of the program in **Figure 6**? Justify your answer.

(3 marks)

```
 1   public class AssertionTest {
 2     public static void main(String[] args) {
 3       try {
 4           int[] array = new int[5];
 5           int i = array[10];
 6       }
 7       catch (Exception ex) {
 8         System.out.println("Caught an exception!");
 9       }
10       catch (Error err) {
11         System.out.println("Caught an error!");
12       }
13     }
14   }
```

Figure 6

	Do not write in this column
i)	**4 marks**
If assertions are disabled, the program outputs nothing [1 mark]. This is because the	
statement **`assert(false);`** is not run and so nothing is caught by either of the	

	Do not write in this column
catch blocks [1 mark].	
If assertions are enabled, the program outputs the text **Caught an error!**	
[1 mark].	
Enabling assertions forces the statement **assert(false);** to be run [0.5 mark];	
because the assertion expression evaluates to **false**, this triggers an	
AssertionError (which is a sub-class of **Error**) to be thrown [0.5 mark].	
ii)	**3 marks**
The program's output will be the text **Caught an exception!** [1 mark] because	
the code in the **try** block fails [0.5 mark]; the **int** array only has 5 elements in it,	
yet the next line tries to access the 10th element in the array [0.5 mark].	
The method then throws an exception of type	
ArrayIndexOutOfBoundsException [0.5 mark] which is caught by the **catch**	
block [0.5 mark].	
	7 marks

b) Consider the Java statement in **Figure 7** and answer the following questions:

[4 marks]

```
int[][] matrix = new int[5][5];
```

Figure 7

i) Write a Java statement that stores the value **200** in the second row of the third column of the array.

(1 mark)

ii) Write Java code to store the numbers **1 2 3 4 5** in the first row, numbers **6 7 8 9 10** in the second row, and so on until **21 22 23 24 25** in the last row.

Note: You are not required to write a complete Java program.

(3 marks)

	Do not write in this column
i)	**1 mark**
`matrix[1][2] = 200;` [1 mark]	
ii)	**3 marks**
`for (int i=0; i<5; i++) {` [0.5 mark]	
`  for (int j=0; j<5; j++) {` [0.5 mark]	
`    matrix[i][j] = (j+1) + (5*i);` [2 marks]	
`  }`	
`}`	
(Note: Other sensible solutions are accepted.)	
	4 marks

c) Answer the following questions about File Input/Output in Java:

[9 marks]

i) Java I/O is usually defined using *streams*. Briefly describe the concept of *streams* in Java.

(1.5 marks)

ii) Write a Java program called **`PriceWriter`**. This class is responsible for writing a list of prices into a file. This class must create a file named **`prices.dat`** and write a list of prices (e.g. **`90`**, **`5`**, **`20`**) into the file. Each price should be written to the file on a separate line preceded by a dollar (**`$`**) sign. **Figure 8** gives an example of the resulting **`prices.dat`** file's contents:

```
$90
$5
$20
```

Figure 8

You can assume that all the prices are available in an array of integer values (you must declare this array in your own code and add some example prices into it). The file should be created in the same directory as the program. If the file already exists, then the program must display the message "**`File already exists`**" and then terminate the program.

(7.5 marks)

	Do not write in this column
i)	**1.5 marks**
A stream is a connection to a source of data or to a destination for data (sometimes both) [1 mark]. Streams can represent any data, so a stream is a sequence of bytes that flow from a source to a destination [0.5 mark].	
ii)	**7.5 marks**

```
import java.io.*;                               [0.5 mark]
public class PriceWriter {                      [0.5 mark]
  public static void main(String[] args) {
    int[] prices = {100,45,83,42};
    File myFile = new File("prices.dat");    [1 mark]
    if (myFile.exists()) {
      System.out.println("File already exists."); [1 mark:
      System.exit(1);                              sensible attempt]
    }
    try {
      FileWriter output = new FileWriter("prices.dat"); [1 mark]
      for (int i=0; i<prices.length; i++)
        output.write("$" + marks[i] + "\n");          [1 mark]
      output.close();                                  [1 mark]
    }
    catch (IOException ex) { ex.printStackTrace(); } [1 mark]
  }
} [0.5 mark: sensible program structure]
```

		9 marks

d) This question is about Graphical User Interfaces in Java.

Write a block of Java code that creates a **`JButton`** and places it inside of a **`JFrame`**. The **`JButton`** should contain the text "**`Hello`**". Add appropriate *event handing* code so that the text changes to "**`Bye Bye`**" after somebody clicks the button.

Note: You are not required to write a complete Java program.

[5 marks]

	Do not write in this column
`JFrame myFrame = new JFrame();` [0.5 mark]	
`JButton myButton = new JButton("Hello");` [0.5 mark]	
`myFrame.getContentPane().add(myButton);` [1 mark]	
`myButton.addActionListener(new ActionListener()` [1 mark]	
`{`	
`  public void actionPerformed(ActionEvent e) {` [1 mark]	
`    myButton.setText("Bye Bye");` [1 mark]	
`  }`	
`});`	
(Note: It is also acceptable to define the **`ActionListener`** in a separate class.)	
	5 marks

Question marking: $\frac{}{7} + \frac{}{4} + \frac{}{9} + \frac{}{5} = \frac{}{25}$