

**Q1.** Write a Java class called Circle. A Circle has a centre point (stored as x and y values) and has a radius.

- Include a constructor that allows a new Circle object to be created with given centre point and radius values.
- Include a method to your class that returns the area of a Circle object.
- Add a main() method to your class which creates an array of three Circle objects. In main(), add a for loop to print out the area of each Circle object in the array.

**[15 marks]**

**Q2.** Write a Java class called Module.

- A Module has a module code (e.g. "EE219") and a module name (e.g. "Object Oriented Programming I").
- A Module also has a credit value (e.g. the value 5.0).
- Add a constructor to your class that allows a new Module object to be initialized with a code, name and credit value.
- Add an overridden `String toString(){...}` method so that a Module can be printed using the following test code:

```
public static void main(String args[]) {  
    Module m1 = new Module("EE219", "Object Oriented Programming I", 5.0);  
    System.out.println(m1);  
}
```

Running this code should print out:

```
EE219 Object Oriented Programming I (5.0 credits)
```

**[10 marks]**

**Q3.** Write a Java class called Complex that implements the same functionality as the C++ Complex class in [Assignment 5 solutions](#). You may use and adapt the C++ solution code to write your Java class.

Your Complex class will have:

- A constructor that takes real and imaginary component values to initialize a new Complex object
- The methods `add()`, `subtract()`, `multiply()` and `divide()`. Note that there are no overloaded operators in Java.
- An overridden `String toString(){...}` method, to implement similar functionality as the overloaded `<<` operator in the C++ version of the class. This Java method will return a string representation of the Complex object.

Your class's interface will be implemented so that the following test code:

```
public static void main(String args[]) {  
    Complex c1 = new Complex(1,-2);  
    Complex c2 = new Complex(3,-4);  
    Complex c3 = new Complex(-3,2);  
    System.out.println(c1.add(c2).subtract(c2).subtract(c1));  
    System.out.println(c1.subtract(c2).divide(c3).add(c1.multiply(c3)));  
}
```

will run and print out:

```
0.0
```

```
1.7692307692307692 + 7.846153846153846j
```

**[20 marks]**

**Q4.** Consider the following abstract Student class:

```
public abstract class Student {
    private String name;
    private int ID;
    public Student(String nm, int id) {
        name = nm;
        ID = id;
    }
    void print() {
        System.out.println("Name: " + name + "\nID: " + ID);
    }
}
```

Write a Java class called UndergradStudent that inherits from the Student class. An UndergradStudent object will have:

- A String member to store the student's programme (e.g. "ME")
- An array of Modules, using your Module class implementation for Q2. The array should be able to store up to 10 modules.
- A data member numModules that keeps track of the current number of modules stored in an UndergradStudent object. When initially created, an UndergradStudent stores no modules.
- A method boolean add(Module mod), which adds a new module to the array of modules stored in an UndergradStudent object. The method returns false if the modules array is full, and returns true otherwise.
- A suitable constructor to initialize a new UndergradStudent object
- A void print() method which prints out all information for an UndergradStudent. The method should use the print() method of the inherited Student class to print the name and ID.

Include the following test main() method in your class:

```
public static void main(String args[]) {
    UndergradStudent s1 = new UndergradStudent("Mary Murphy", 12345678, "ECE");
    s1.add(new Module("EE219", "Object Oriented Programming I", 5.0));
    s1.add(new Module("EE223", "Digital and Analogue Electronics I", 5.0));
    s1.add(new Module("EE202", "Embedded Systems", 5.0));
    s1.print();
}
```

which should result in the print out:

```
Name: Mary Murphy
ID: 12345678
Programme: ECE
Registered for modules:
    EE219 Object Oriented Programming I (5.0 credits)
    EE223 Digital and Analogue Electronics I (5.0 credits)
    EE202 Embedded Systems (5.0 credits)
```

**[20 marks]**

**Q5.** Write a recursive method in Java void mirrorString(String str, int i){...} which takes the string str and prints out the string followed by it's mirror image. For example, when called using mirrorString("Hello World!"); it will print the string "Hello World!!dlrow olleH".

The following skeleton class is provided for developing your method. Submit only the code for your completed method.

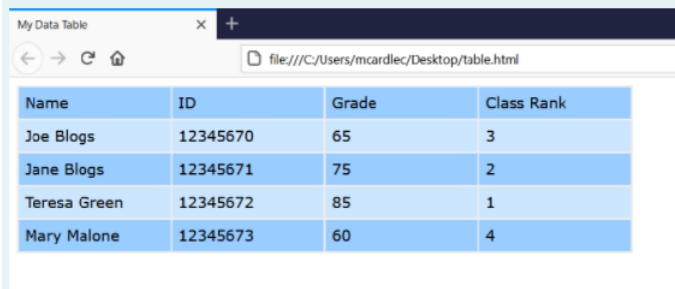
```
public class MirrorString {
    public static void mirrorString(String str) {
        mirrorString(str, 0);
    }
    private static void mirrorString(String str, int i) {

        // implement this method

    }
    public static void main(String args[]) {
        mirrorString("Hello World!");
    }
}
```

**[10 marks]**

**Q6.** Write a Java class called `HTMLTableGenerator` that allows a table of data to be created and then printed out to the console in the form of a valid HTML table. When the print out is saved to a text file named `table.html` and opened in a web browser, something similar to the following should be displayed:



Name	ID	Grade	Class Rank
Joe Blogs	12345670	65	3
Jane Blogs	12345671	75	2
Teresa Green	12345672	85	1
Mary Malone	12345673	60	4

Your `HTMLTableGenerator` class will have the following methods:

- A constructor that takes two int values that specify the number of rows and the number of columns that the table will have.
- A `void addRow(String fields[])` method, which takes an array of strings and adds the strings as a new row of data in the table stored in the `HTMLTableGenerator` object.
- A `void print()` method, which prints out the table in HTML form. See: [https://www.w3schools.com/html/html\\_tables.asp](https://www.w3schools.com/html/html_tables.asp)

**Hint:** Store your table data as a 2D array of strings: `private String table[][]`. `table[r][c]` is then the string in row `r` and column `c`.

The following test code should produce a table similar to above:

```
public static void main(String args[]) {
    HTMLTableGenerator table = new HTMLTableGenerator(5,4);
    table.addRow(new String[] { "Name",      "ID",      "Grade", "Class Rank" } );
    table.addRow(new String[] { "Joe Blogs",  "12345670", "65"    , "3"        } );
    table.addRow(new String[] { "Jane Blogs",  "12345671", "75"    , "2"        } );
    table.addRow(new String[] { "Teresa Green", "12345672", "85"    , "1"        } );
    table.addRow(new String[] { "Mary Malone", "12345673", "60"    , "4"        } );
    table.print();
}
```

You may choose your own fonts, colors and styling for your table.

**[25 marks]**