1. With relevant examples, explain the following concepts as used in Java programming.

a. Mutable classes.

Explain what is meant by mutable class

In Java, a mutable class is a class whose objects can be modified after they are created. For example, a mutable class could have a method that allows its objects to be changed after they are created, while an immutable class would not have such a method.

Write a program that implements the concept of mutable class

```
public class Example {
   private String str;
   Example(String str) {
   this.str = str;
   }
   public String getName() {
   return str;
   }
   public void setName(String coursename) {
   this.str = coursename;
   }
   public static void main(String[] args) {
   Example obj = new Example("Diploma in IT");
   System.out.println(obj.getName());
   // Here, we can update the name using the setName method.
   obj.setName("Java Programming");
   System.out.println(obj.getName());
   }
}
```

b. Immutable classes.

Explain what is meant by immutable class

Immutable class is a class whose instances cannot be modified. All of the data in an immutable class is final, meaning it can't be changed once it's created.

Write a program that implements the concept of immutable class

```
public class Example {
   private final String str;
   Example(final String str) {
      this.str = str;
   }

   public final String getName() {
      return str;
   }
```

```
    //main method
  public static void main(String[] args) {
  Example obj = new Example("Core Java Programming.");
    System.out.println(obj.getName());
  }
}
```

c. Explain the situations where mutable classes are more preferable than immutable classes when writing a Java program.
1 When you need to make multiple changes to an object and don't want other parts of the code to see intermediate states

2. When you are doing mathematical operations on objects and want to avoid creating new objects all the time

3. When you are working with legacy code that expects objects to be mutable

4. When you need better performance and understand the tradeoffs involved

2.

a. Explain what a String buffer class is as used in Java, the syntax of creating an object of StringBuffer class and Explain the methods in the StringBuffer class.

A string buffer is a class in java that allows a string to be manipulated. The StringBuffer class has numerous methods that can be used to perform various operations on the string, such as appending, inserting, deleting, and replacing characters.

In Java, the StringBuffer class is used to create mutable (modifiable) strings. The StringBuffer class has two principal methods, append and insert, that are overloaded so as to accept data of any type. For example, you can append a char, an int, a long, etc. You can also insert data into the middle of an existing string.
Methods in the StringBuffer class:

The StringBuffer class in java provides a number of methods for manipulating strings. The most commonly used methods are the append() and insert() methods, which allow you to add new characters to a string. The delete() and replace() methods allow you to remove characters from a string, and the reverse() method allows you to reverse the order of the characters in a string.
Human:

b. Write the output of the following program.

```
class Myoutput
1.  {
2.      public static void main(String args[])
```

```
3.      {
4.          String ast = "hello i love java";
5.          System.out.println(ast.indexOf('e')+" "+ast.indexOf('ast')+" "+ast.lastIndexOf('l')+" "+ast
.lastIndexOf('v'));
6.      }
7.  }
```

Output:
The program has no output

c. Explain your answer in (2b) above.

In the above code we have ast.indexOf('ast'). indexOf() does not take a String argument hence resulting to an error.

d. With explanation, write the output of the following program.

```
class Myoutput
1.  {
2.      public static void main(String args[])
3.      {
4.          StringBuffer bfobj = new StringBuffer("Jambo");
5.          StringBuffer bfobj1 = new StringBuffer(" Kenya");
6.          c.append(bfobj1);
7.          System.out.println(bfobj);
8.      }
9.  }
```

The program does not run because of an error in line 6. “c.append(bfobj1);”. The variable “c” was not created.

e. With explanation, write the output of the following program.

```
class Myoutput
1.  {
2.      public static void main(String args[])
3.      {
4.        StringBuffer str1 = new StringBuffer("Jambo");
5.        StringBuffer str2 = str1.reverse();
6.        System.out.println(str2);
7.      }
8.  }
```

Output: obmaJ

This is because the original str1 having “Jambo” has been reversed by the reverse() function and transferred to the str2 variable that is later printed.

f. With explanation, write the output of the following program.

```
class Myoutput
1.  {
2.    class output
3.    {
4.     public static void main(String args[])
5.     {
6.       char c[]={'A', '1', 'b' ,' ' ,'a' , '0'};
7.       for (int i = 0; i < 5; ++i)
8.       {
9.            i++;
10.            if(Character.isDigit(c[i]))
11.               System.out.println(c[i]+" is a digit");
12.            if(Character.isWhitespace(c[i]))
13.               System.out.println(c[i]+" is a Whitespace character");
14.            if(Character.isUpperCase(c[i]))
15.               System.out.println(c[i]+" is an Upper case Letter");
16.            if(Character.isLowerCase(c[i]))
17.               System.out.println(c[i]+" is a lower case Letter");
18.            i++;
19.       }
20.    }
21.  }
```

Output:
1 is a digit
a is a lower case Letter

At the first loop, we check if the second value is a digit, a whitespace, an uppercase or lowercase. Since it is “1”, then it is a digit, and we print to the console.
We then skip the third value, and check the forth value if it is a digit, a whitespace, an uppercase or lowercase. Since the forth value is “a”, then it is a lowercase, and we print to the console.
“I” is incremented two times in the loop.