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.

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.
There are a few situations where mutable classes are more preferable than immutable classes:

1. When you need to change the value of an object after it is created, a mutable class is more suitable. For example, if you want to increment a counter variable, a mutable class would be more appropriate.

2. Mutable classes can be more efficient in some cases, since they don't need to create a new object each time the value is changed.

3. Mutable classes can be easier to work with in some situations. For example, if you need to update a list of objects, a mutable class can make this task simpler.

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 mutable sequence of characters. Unlike a string, a string buffer can be modified after it has been created. The StringBuffer class in Java provides methods for appending, inserting, and replacing characters in a string buffer.

The syntax of creating a StringBuffer object is: StringBuffer buffer = new StringBuffer();

There are a number of methods in the StringBuffer class that can be used to manipulate strings, including:
append() - Adds new content to the end of the buffer
insert() - Adds new content at a specified position within the buffer
delete() - Deletes a specified range of characters from the buffer
replace() - Replaces a specified range of characters in the buffer with new characters
reverse() - Reverses the order of the characters in the buffer
toString() - Returns the contents of the buffer as a string

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.