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

Explain what is meant by mutable class

- A mutable class is a class whose objects can be modified after they are created. In other words, the state or contents of a mutable object can be changed after it is constructed.

Write a program that implements the concept of mutable class

```
public class MutableExample {
  private String value;

  public MutableExample(String value) {
    this.value = value;
  }

  public String getValue() {
    return value;
  }

  public void setValue(String value) {
    this.value = value;
  }
}
```

b. Immutable classes.

Explain what is meant by immutable class

- An immutable class is a class whose objects cannot be modified after they are created. In other words, the state or contents of an immutable object cannot be changed after it is constructed.

Write a program that implements the concept of immutable class

```
public final class ImmutableExample {
  private final String value;

  public ImmutableExample(String value) {
    this.value = value;
  }

  public String getValue() {
    return value;
  }
}
```

c. Explain the situations where mutable classes are more preferable than immutable classes when writing a Java program.

- Performance: Mutable classes can often be more efficient, because they do not need to create a new object every time the object's state needs to be changed. For example, if you have a large object with many fields that need to be modified frequently, using a mutable class might be more efficient than using an immutable class.
-

- Flexibility: Mutable classes can be more flexible, because they allow you to modify the object's state after it is created. This can be useful if you need to change the object's state in response to changing conditions or user input.
-
- Complex objects: Mutable classes can be easier to use when creating complex objects with many fields and relationships, because you can set the fields and relationships as needed rather than having to create a new object every time the object's state changes.

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.

- The StringBuffer class is a mutable sequence of characters that can be modified after it is created. It is similar to the String class, but allows you to modify the contents of the string without creating a new object.

```
StringBuffer sb = new StringBuffer(); // Creates an empty StringBuffer
StringBuffer sb = new StringBuffer(int capacity); // Creates a StringBuffer with the specified capacity
StringBuffer sb = new StringBuffer(String str); // Creates a StringBuffer with the specified string
```

The StringBuffer class has several methods for modifying the contents of the string:

- append: Appends the specified string or object to the end of the string buffer.
- insert: Inserts the specified string or object at the specified position in the string buffer.
- replace: Replaces the specified range of characters in the string buffer with the specified string.
- delete: Deletes the specified range of characters from the string buffer.
- reverse: Reverses the order of the characters in the string buffer.

b) Write the output of the following program.

```
class Myoutput

{
public static void main(String args[])
{

String ast = "hello i love java";

System.out.println(ast.indexOf('e')+" "+ast.indexOf('ast')+"
"+ast.lastIndexOf('l')+" "+ast .lastIndexOf('v'));
}
}
```

1 -1 9 11

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

- The indexOf method returns the index of the first occurrence of the specified character or string in the original string, or -1 if the character or string is not found. The lastIndexOf method returns the index of the last occurrence of the specified character or string in the original string, or -1 if the character or string is not found.

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

```
class Myoutput

{
```

```
public static void main(String args[])
{
StringBuffer bfobj = new StringBuffer("Jambo");
StringBuffer bfobj1 = new StringBuffer(" Kenya");
c.append(bfobj1);
System.out.println(bfobj);
}
}
```

Jambo Kenya

- The StringBuffer class is a mutable sequence of characters that can be modified after it is created. In this case, the program creates two StringBuffer objects, bfobj and bfobj1, with the strings "Jambo" and " Kenya" respectively.

- Then, the program calls the append method on bfobj, passing in bfobj1 as an argument. The append method appends the contents of bfobj1 to the end of bfobj. After this operation, bfobj contains the string "Jambo Kenya".

- Finally, the program prints out the contents of bfobj using the println method. This outputs the string "Jambo Kenya" to the console.

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

```
class Myoutput
{
public static void main(String args[])
{
StringBuffer str1 = new StringBuffer("Jambo");
StringBuffer str2 = str1.reverse();
System.out.println(str2);
}
}
```

obmaJ

- The StringBuffer class is a mutable sequence of characters that can be modified after it is created. In this case, the program creates a StringBuffer object called str1 with the string "Jambo".

- Then, the program calls the reverse method on str1, which reverses the order of the characters in the string buffer. After this operation, str1 contains the string "obmaJ".

- The reverse method returns a reference to the modified StringBuffer object, so the program assigns this reference to the str2 variable.

- Finally, the program prints out the contents of str2 using the println method. Since str2 refers to the same object as str1, this outputs the reversed string "obmaJ" to the console.

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

```
class Myoutput

{
class output
{
public static void main(String args[])
{
char c[]={'A', '1', 'b' ,' ' ,'a' , '0'};
for (int i = 0; i < 5; ++i)
{
i++;
if(Character.isDigit(c[i]))
System.out.println(c[i]+" is a digit");
if(Character.isWhitespace(c[i]))
System.out.println(c[i]+" is a Whitespace character");
if(Character.isUpperCase(c[i]))
System.out.println(c[i]+" is an Upper case Letter");
if(Character.isLowerCase(c[i]))
System.out.println(c[i]+" is a lower case Letter");
i++;
}
}
}
```

A is an Upper case Letter
b is a lower case Letter
a is a lower case Letter
0 is a digit

- The Character class in Java provides a set of methods for determining the characteristics of individual characters. In this case, the program creates an array of characters called c, containing the elements 'A', '1', 'b', ' ', 'a', and '0'.

- The program then iterates over the elements of c, using a for loop. For each iteration, the loop variable i is incremented by 2. This means that the loop will only process the elements at the odd indices of the array (0, 2, 4).

- For each element in the array, the program checks if it is a digit, a whitespace character, an upper case letter, or a lower case letter using the isDigit, isWhitespace, isUpperCase, and isLowerCase methods of the Character class. If any of these conditions are true, the program prints out a message indicating which condition is satisfied.

- Since the elements of the array are 'A', 'b', 'a', and '0', the program will print out messages indicating that 'A' is an upper case letter, 'b' is a lower case letter, 'a' is a lower case letter, and '0' is a digit.