Part 1

1.1 (3 points)

Given the following program, which data structure results in the main-method afte line 19 is executed?

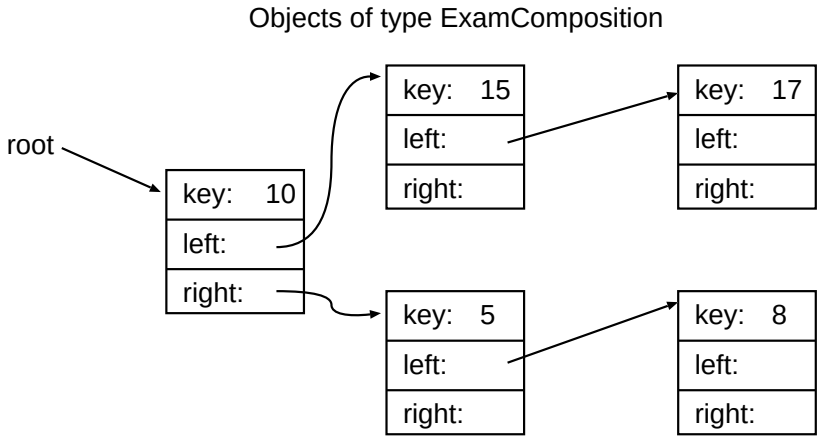
Complete the given sketch by drawing the missing references and the values stored in the member-variables `key`.

```
package composition;
public class ExamComposition {
    int key;
    ExamComposition left;
    ExamComposition right;

    ExamComposition(int key_) {
        key = key_;
    }

    public static void main(String[] args) {
        ExamComposition root = new ExamComposition(10);
        root.left = new ExamComposition(15);
        root.right = new ExamComposition(5);
        ExamComposition tmpComp = root.right;
        tmpComp.left = new ExamComposition(8);
        tmpComp = root.left;
        tmpComp.left = new ExamComposition(17);
        System.out.println(". . . done");
        return;
    }
}
```

Solution:



1.2 (3 Points)

What is the output of the given program?

```
public class DemoBaseClass {
    protected int count;
    DemoBaseClass() {
        count = 99;
    }
    DemoBaseClass(int count_) {
        count = count_;
    }
    public void printDemo() {
        System.out.println("count: " + count);
    }
    public static void main(String[] args){
        DemoBaseClass myObject = new DemoBaseClass(42);
        myObject.printDemo();
        myObject = new DemoSubClass1();
        myObject.printDemo();
        myObject = new DemoSubClass2();
        myObject.printDemo();
    }
}

public class DemoSubClass1 extends DemoBaseClass{
    int value = 10;
    DemoSubClass1(){
    }
    public void printDemo() {
        int tmp = count + value;
        System.out.println("count + value: " + tmp);
    }
}

public class DemoSubClass2 extends DemoBaseClass{
    int value;
    DemoSubClass2(){
        super(33);
        value=44;
    }
}
```

Complete the the output:

```
count: 42
count + value: 109
count: 33
```

Part 2

2.1 Exception Handling

A program defines two exception classes and a main class:

```
public class TestException extends Exception{
}
```

```
public class RareException extends TestException {
    float exValue;
    RareException(float value_) {
        exValue = value_;
    }
}
```

```
public class ExceptionProgram {
    public ExceptionProgram() {
    }

    public static void main(String[] args) {
        ExceptionProgram exProgObj = new ExceptionProgram();
        exProgObj.testEx(exProgObj, 2);
        exProgObj.testEx(null, 4);
        exProgObj.testEx(exProgObj, -3);
        exProgObj.testEx(exProgObj, 0);
        return;
    }

 void testEx(ExceptionProgram exProgram, int number) {
        // A
        try {
            int result = checkExProg(exProgram, number);
            System.out.println("checkExProg, result: " + result);
        }
        // B
        catch (RareException rareEx) {
            System.out.println(" --> cought RareException:" + rareEx.exValue);
        }
        // C
        catch (Exception ex) {
            System.out.println(" -- > cought general Exception");
        }
        // D
        return;
    }
```

```
    int checkExProg(ExceptionProgram exProgram, int number) throws Exception {
        // E
        System.out.println("value: " + exProgram.value);
        // F
        if (number <= -1) {
            throw new RareException(number);
        }
        // G
        return value / number;
        // H
    }
}
```

2.2 (3 Points)

Define a new exception class named `DerivedException` as a subclass of `TestException`. The class `DerivedException` defines neither a member nor a constructor.

```
public class DerivedException extends TestException{
}
```

2.3 (3 Points)

Insert code into the method `checkExProg()` so that an object of the newly defined exception is thrown if the parameter `exProgram` has the value `null`.

First just provide the code for throwing the exception:

```
if (exProgram == null) {
    throw new DerivedException();
}
```

Now please indicate where this code goes. The exception throwing code could/should replace (X all applicable answers):

- ⊗ // E
- ○ // F
- ○ // G
- ○ // H

2.4 (x Points)

Insert code into the method `foo()` so that the newly thrown exception is cought. In the catch block print out: `-> cought DerivedException`.

First just provide the code for catching the exception:

```
catch (DerivedException drEx) {
    System.out.println(" --> cought DerivedException");
}
```

Now please indicate where this code goes. The exception catching code could/should replace (X all applicable answers):

- ◯ // A
- ⊗ // B
- ⊗ // C
- ◯ // D

Part 3

3.1 (x points) File IO

Given is the following excerpt of a Java programm:

The class `DataHolder` encloses three members:

```
public class DataHolder implements Serializable {
    public int dataID;
    public double temperature;
    public String name;
    . . .
```

The class `Java_IO_Example` contains the following members and methods

```
public class Java_IO_Example {
   DataHolder[] dataObjects = new DataHolder[10];
   . . .
   void saveDataHoldersDataStream() {
        BufferedOutputStream bos = getBufferedOutputStream(dataStreamFile);
        DataOutputStream dos = null;
        try {
            dos = new DataOutputStream(bos);
            for (int i = 0; i < dataObjects.length; i++) {
                saveOneDataHolderDataStream(dataObjects[i], dos);
            }
        } catch (Exception ex) {
            ex.printStackTrace();
        }
        try {
            dos.close();
        } catch (Exception ex) {
            ex.printStackTrace();
        }
    }

    void saveOneDataHolderDataStream(DataHolder dataObj, DataOutputStream dos
       . . .
    }

    BufferedOutputStream getBufferedOutputStream(String fileName) {
        . . .
    }
```

The method `getBufferedOutputStream()` creates a buffered InputStream which is connected to a file. If the method can not return a stream it stops the program calling `System.exit(1)`. Fill in the missing code:

```
BufferedOutputStream getBufferedOutputStream(String fileName) {
    FileOutputStream fos = null;
    BufferedOutputStream bos = null;
    try {
        fos = new FileOutputStream(fileName);
        bos = new BufferedOutputStream(fos);
    } catch (Exception ex) {
        ex.printStackTrace();
    }
    if (bos == null) {
        System.out.println("could not open file - stopping program");
        System.exit(1);
    }
    return bos;
}
```

The method `saveOneDataHolderDataStream()` writes all members of the object `dataObj` to the stream `dos`. Fill in the missing code:

```
void saveOneDataHolderDataStream(DataHolder dataObj, DataOutputStream dos) {
    try {
        dos.writeInt(dataObj.dataID);
        dos.writeDouble(dataObj.temperature);
        dos.writeUTF(dataObj.name);

    } catch (Exception ex) {
        ex.printStackTrace();
        System.out.print("could not save dataHolder: ");
        System.out.println(dataObj.name);
    }
}
```

The method `readOneDataHolderDataStream()` reads three values (data type `int`, `double`, `String`) from the stream `dis` and creates an object of type `DataHolder` from these values. It returns the object to the caller.

```
DataHolder readOneDataHolderDataStream(DataInputStream dis)
            throws IOException {
    DataHolder dataObj = null;
    int dataId  = dis.readInt();
    double temp = dis.readDouble();
    String name = dis.readUTF();
    dataObj = new DataHolder(dataId, temp, name);
    return dataObj;
}
```