Final

Name_________________________ **CSC110 Fall 2019** **Professor Conroy**

(333 total points)

Trace (3pts each / 108pts total)

Write the output of the following program fragments: OUTPUT:

```
1.System.out.print(“B\na\\nana”);
```

```
2.
System.out.println(“You do”);
System.out.println(“ say!”);
```

```
3.
System.out.print(“\”RSA Encrypt\””);
```

```
4.
System.out.print(“Fox”);
System.out.print(“Clocks”);
```

5.
```
boolean scala = false;
boolean julia = true;

if(scala || julia){
   System.out.println("Broad City");
}

if(!false){
  System.out.println("Mandalorian");
}
```

6.
```
boolean webassembly = false;
boolean webcomponents = true;

if(webassembly && webcomponents){
  System.out.println("Rick and Morty");
} else{
  System.out.println("Party Down");
}
```

7.
```
boolean rust = false;
boolean go = true;

if(rust || go){
  System.out.println("The Irishmen");
}

System.out.println("The Watchman");
```

```
8.
boolean python = true;

if(7 < 9 && python){
   System.out.println(“6”);
   if(!(false || false)){
     System.out.println(“Feet”);
   } else{
      System.out.println(“Under”);
   }
} else {
   System.out.println(“Mrs. Maisel”);
}
```

```
9.
int php = 3;
int perl = php;

php = php + 1;

System.out.println(php);
System.out.println(perl);
```

```
10.
String planet = “Earth”;

switch(planet){
  case “Mercury”:
    System.out.println(“The Favorite”);
    break;
   case “Earth”:
     System.out.println(“PEN15”);
     break;
    case “Jupiter”:
      System.out.println(“Mad Men”);
    case “Uranus”:
       System.out.println(“The Americans”);
}
```

11.
```
if(‘c’ != ‘c’){
  System.out.println(“Roma”);
} else if (42 > 2){
  System.out.println(“Greenbook”);
} else{
  System.out.println(“Parasite”);
}
```

12.
```
Circle bash = new Circle();
bash.setRadius(16);

Circle cshell = bash;

cshell.setRadius(9);

System.out.println(bash.getRadius());
System.out.println(cshell.getRadius());
```

13.
```
int unix = 88;
unix++;
System.out.println(unix);
```

14.
```
int bitcoin = 13;
int ethereum = 2;

System.out.println(bitcoin / ethereum);
```

15.
```
int nodejs = 73;
nodejs += 2;
System.out.println(nodejs);
```

16.
```
int clojure = 99;
clojure--;
System.out.println(clojure);
```

17.
```
int unity3d = 26;
int godotengine = 2;
int unreal = -1;

System.out.println(unity3d % godotengine);
```

18.
```
double csharp =  7;
double cplusplus = 2;

System.out.println(csharp / cplusplus);
```

19.
```
int sqllite = Math.pow(2,6);
System.out.println(sqllite);
```

20.
```
int max = Integer.MAX_VALUE; //2147483647, the largest value of type int
int min = Integer.MIN_VALUE; //-2147483648, the smallest value of type int

min--;

System.out.println(min);
```

21.
```
char vuejs = 67;  //the letter C
vuejs--;

System.out.println(vuejs);
```

22.
```
int tensorflow = 3 + 8 * 4 / 8 + 2;
System.out.println(tensorflow);
```

23.
```
int linux = (3 + 1) * (11 + 1) * 2
System.out.println(linux);
```

24.
```
String websocket = “1”;
String nmap = “2”;
String siege = “3”;

System.out.println(websocket + nmap + siege);
```

25.
```
String tcp = “  Downton  “;
String ip = “Abbey”;

System.out.println(tcp.toLowerCase().trim() + ip);
```

26.
```
int virtualreality = 2;
int augmentedreality = 5;

System.out.println((double) virtualreality / (double) augmentedreality);
```

27.
```
String interplanetaryfilesystem = “42”;
String diaspora = “22”;

System.out.println(Integer.parseInt(interplanetaryfilesystem) + Integer.parseInt(diaspora));
```

28.
```
String meshnetwork = “”;

System.out.println(“The Godfather”);

do{
System.out.println(“Sticks and Stones”);
meshnetwork += “Sorry to Bother You.”;
}while (!meshnetwork.equals(“Sorry to Bother You.”));

System.out.println(“The Crown”);
```

29.
```
String fediverse = “Exit through the Gift Shop”;

System.out.println(“Stranger Things”);

while(fediverse == “The Revenant”){
  System.out.println(“As Good as it Gets”);
}

System.out.println(“Rainman”);
```

30.
```
String deeplearning = “”;

System.out.println(“Birdman”);

for(int i=0;i < 3; i++){
  for(int j=0;j < 2; j++){
    deeplearning += “The 100”;
  }
  System.out.println(i + “ “ + deepLearning);
}

System.out.println(deeplearning);
```

31.
```
String activitypub = “”;

System.out.println(“Whiplash”);

int ostatus = 1;

while(!activitypub.equals(“aaaa”)){
  System.out.println(“Fleabag” + ostatus);

  for(int i=0; i<2; i++){
    System.out.println(“ostatus“ + ostatus + “ loop iteration “ + i);
    activitypub += “a”;
  }

  ostatus++;
}

System.out.println(“Russian Doll”);
```

32. (3pts)

```
public class Trace32 {
  public static void main(String [] args){
    Trace32 westwing = new Trace32(5);
    System.out.println(westwing.getCommunity());
    westwing.increment();
    System.out.println(westwing.toString());
  }

  public Trace32(int firefly){
    community = firefly;
  }

  public int getCommunity(){
    return community;
  }

  public void increment(){
    community++;
  }

  public String toString(){
    return Integer.toString(getCommunity());
  }

  private int community;
}
```

33. (3pts)

```
public class Trace33 {
  public static void main(String [] args){
    String t = "Sorry to Bother You";
    String s = "Succession";
    String u = “Westworld”;

    System.out.println(Trace33.concat(t, s));

    Trace33 t33 = new Trace33(t, u);

    System.out.println(t33.concat(s));
  }

  public Trace33(String x, String y){
   str = x;
   separator = y;
  }

  public static String concat(String a, String b){
    return a + b;
  }

  public String concat(String a){
   return this + separator + a;
  }

  public String toString(){
    return str;
  }

  private String str;
  private String separator;
}
```

34. (5pts)

```
public class Trace34{
  public static void main(String [] args){
    System.out.println(build("ab"));
  }

  public static String build(String s)
  {
    if (s.length() > 10){
        System.out.println(s);
        return s;
    }else{
        System.out.println(s);
        String t = s;
        return build(t + t);
    }
  }
}
```

35.
```
import java.io.IOException;
import java.io.File;
import java.io.PrintWriter;

public class Trace35{
  public static void main(String [] args) throws IOException{
    PrintWriter x = new PrintWriter(new File("output.txt"));
    x.print("Remember when you were young, \nyou shone like the sun \n");
    x.print("Shine! On! You crazy diamond!");
    x.close();                                      //output.txt
  }
}
```

36.
```
/* Describe the possible outputs for this code */

public class Trace36{
  public static void main(String [] args){
    System.out.println(Math.ceil(Math.random() * 3));
  }
}
```

Recognize Elements

(5pts each / 55pts total)

Use the code provided as a separate hand-out for the Television program to answer the questions below.

1. List all of the classes included in this program.

2. Which class has a main function?

3. What is the return type of the method getCurrentChannel() in the Television class?

4. Name the public methods (not including constructors) that are in the Television class.

5. Write out the signature of the Remote constructor.

6. What are the parameters of the programButton() method in the Remote class?

7. Name a class and a span of lines (the numbers are to the left) where a for loop is being used.

8. Name a function and the class it belongs to that is responsible for reading data from a file.

9. Name a class and a span of lines where a new Channel object is being instantiated.

10. Name all the member variables of the Television class.

11. Name all of the text files that could provide Channel data for a television.

Conceptual Understanding

(10pts each / 50 pts total + 10pts extra credit)

Use the Television and LivingRoom program provided separately to answer the conceptual questions below:

1. Notice that most of the Television class's methods are private, particularly all of the methods that alter the state of the television (volume, channel, on/off, etc). A Television's state can only be altered by the receiveCommand() method which takes a String that is then converted into a sequence of commands that a dispatchCommand() method uses to issue the private methods. Why would a programmer go through so much trouble to encapsulate the public interface like this? What advantages or disadvantages might such an approach provide?

2. Notice that the Television class owns 5 private member variables of type channel. This illustrates a concept in object-oriented design called composition where part of the outer class's functionality is composed of the state and behavior of the objects it owns. Also notice in the LivingRoom main method there is a variable named tv storing an object of type Television. Note that only two of the tv object's channels are set to a network via its programChannel() method. What will the channels that are set using the programChannel() method output when the tv object calls its receiveBroadcast() method. What will the television output be when streaming from those channels that have not been programmed using programChannel()?

3. Notice in the Television class that there are many public static final variables (constants) declared and that they are used repeatedly throughout the Television class. What would happen if we didn’t use the public static final variables and instead wrote String and numeric literals such as “turnOn”, 1, or 5 throughout the code in all of those various places? What would happen if we needed to change the code so that the value of HIGHEST_VOLUME was 11 instead of 10 while using only numeric and String literals instead of constants? How does this compare to what would need to happen if we needed to update the HIGHEST_VOLUME to 11 as the program is originally written with the constants in place?

4. Notice that the Remote class depends heavily on the existence of the Television class and could not exist without it. When this happens we say that the Remote class is coupled with the Television class. Notice, conversely, that the Television class does not depend on the Remote class. There is no mention of the Remote class in Television and the Television can therefore function without the Remote class. When this happens we say that the Television class is decoupled from or does not depend on the Remote class.

Theoretically, you could invent another class that controls the television apart from the Remote class. To illustrate the concept of decoupling, lets write a new class called Microphone that could be used as a replacement if the Remote class was lost. A user can talk into the microphone to issue commands that change the state of the television. No need to be elaborate, we can just give the Microphone class a single method for turning the tv on or off and for changing the channel up or down. Let the method take a String parameter as input; we can pretend the String was converted from a human voice. Pair a Microphone object to its Television object by passing the Television object into the Microphone class's constructor and setting it to a member variable. You can (and in fact should) rely on the Television class's public methods (for changing the channel and such) while writing the Microphone class's methods.

5. The Channel class imports the classes Scanner, File, and IOException from the packages java.util and java.io respectively to help process files. What is happening code-wise when these classes are being imported? Why are they not just available to your program by default like the System class (of System.out.println)? What advantages are there to forcing programmers to import classes as needed?

6. **EXTRA CREDIT** In what ways could you use arrays to improve the design of the Remote and Television classes. What advantages would a programmer have using this tool?

Use an Existing API

(10pts each / 50pts total)

Pretend you are inside a new main method. Write code for this main method that will execute what is described.

1. Create a new Television and Remote object inside main. Program button 7 of the Remote object to set the volume to the middle level (5). Keep in mind that this will have to work regardless of what volume the Television is currently set at.

2. Create a new Television object and program channel 2 to stream from the network named “cn”

3. Print out the constant GAUSSIAN_NOISE from the Channel class to the screen.

4. Write a function named tivo() that takes a tv as input. The function should receive the television’s current broadcast for 5 iterations and record the data from the broadcast into a file named saved-episode.txt. The function of course is defined outside of main but would be called inside main. You may presume that the appropriate imports have been made and Exceptions have been handled or thrown.

5. Write a function that takes a Television object as input and scans through all of the Television’s channels until it finds the network named “cmt”. Return the channel number once it is found. The function would of course be defined outside main but would be called from within main.

Design a Program

(60pts)

Write an object-oriented model for a casino program. The program will have 3 classes: Customer, SlotMachine, and GoodCasino. The job of the casino is to make customers rich!

The SlotMachine class should have three private member variables of type char.

Each member variable can have one of three possible values:

smiley face	☺	\u263A
heart		\u2764
seven	7	\u0037

The SlotMachine should have an additional member variable of type double named moneyPot that keeps track of how much money the slot machine has.

When a new object of type SlotMachine is constructed without any parameters, it should initialize moneyPot to $1,000,000.00. If a filename is provided to the constructor, the SlotMachine's moneyPot should be initialized to whatever value is stored in the file. You should have two different constructors for either possibility. The filename parameter will be passed in as type String.

The SlotMachine class should have a method named pullLever() that takes an amount of money of type double as input. The pullLever() function will randomly set each of its three char variables to one of the three possible values. If all three char variables are the same (all smileys, all hearts, or all 7's) the slotMachine should reduce its moneyPot by 10 times the amount of money put in the machine (the money passed in to pullLever) and send that amount (10 times the amount put in the machine for the lever pull) as the return variable.

The SlotMachine class should have a toString method that returns the state of all 3 chars as a single String.

The SlotMachine class should have a getMoneyPot() method that returns how much money is left in the machine's moneyPot member variable.

The Customer class should have a private member variable wallet that represents how much money they have. Like the SlotMachine class there should be two constructors, one that takes no parameters but initializes the customer's wallet to $500.00 and another that takes a String representing a filename and loads the amount of money the user has from the value stored in the filename.

The Customer class should have a method spend() that takes an amount of money as input and reduces the amount of money in their wallet by whatever was passed in to the method. So if they have $500 in their wallet and the program calls spend(25), the new value of their wallet will be $475. If the amount of money passed in is greater than the amount of money they have left, the function should set their wallet equal to $0 and return whatever amount of money was remaining in their wallet.

The Customer should also have a receive() method that takes an amount of money as input and adds that amount of money to their wallet. So if receive(50) is called and the user already had $100 in their wallet, the wallet would be updated to have $150.

The Customer should have a checkWallet() function that returns how much money is remaining in their wallet.

Both the Customer and SlotMachine classes should have a save() function that takes a String filename as input and saves the amount of money they have in the file named by the input parameter.

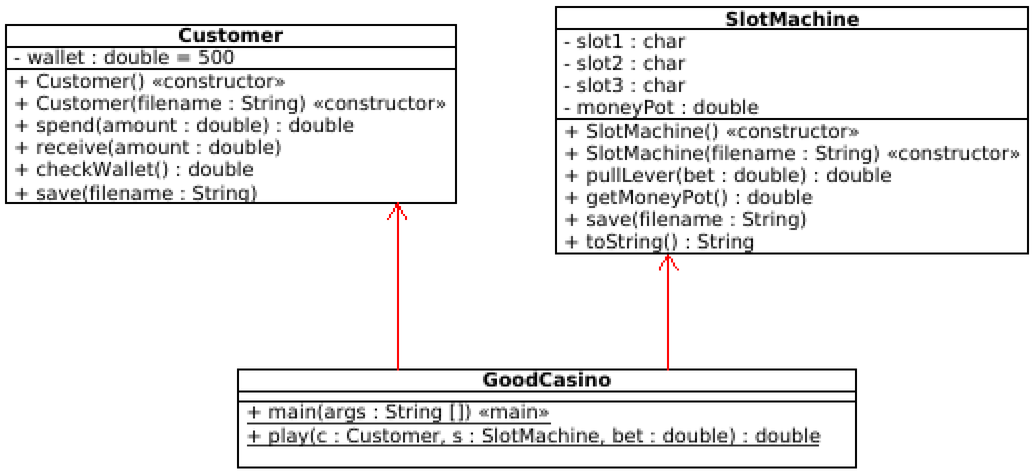
The GoodCasino class should consist of a single main function that creates a new Customer object from the file customer.txt and a new SlotMachine object from the file slot-machine.txt relying on their respective constructors to do the file parsing and variable assignment.

The GoodCasino should have a single static method named play() that takes a Customer, SlotMachine, and amount of money as input. Inside the method, the customer should spend() the amount of money passed in. The amount of money returned by spend() should be passed into the slot machine's pullLever() function. (pullLever() would be called after the call to spend() finishes of course). The play() method should return the amount of money that was returned by the pullLever() method.

The GoodCasino main method should have a loop that asks the user how much money they would like to put into the machine. The play() method should be called with that amount of money. The state of the slot machine chars should be printed to the screen along with the amount of money returned. The amount of money returned should also be added to the customer's wallet. The process should then loop back to the top and ask the user how much money they would like to put in the machine and the whole process should repeat. This should continue until the user or casino runs out of money or until the user types quit. After the loop is finished, the program should save the amount of money stored in the customer's wallet into the file named customer.txt. The program should also save the amount of money stored in the slot machine's money pot into the file named slot-machine.txt.

Please provide an example customer.txt file and an example slot-machine.txt file so I can see your file's format.

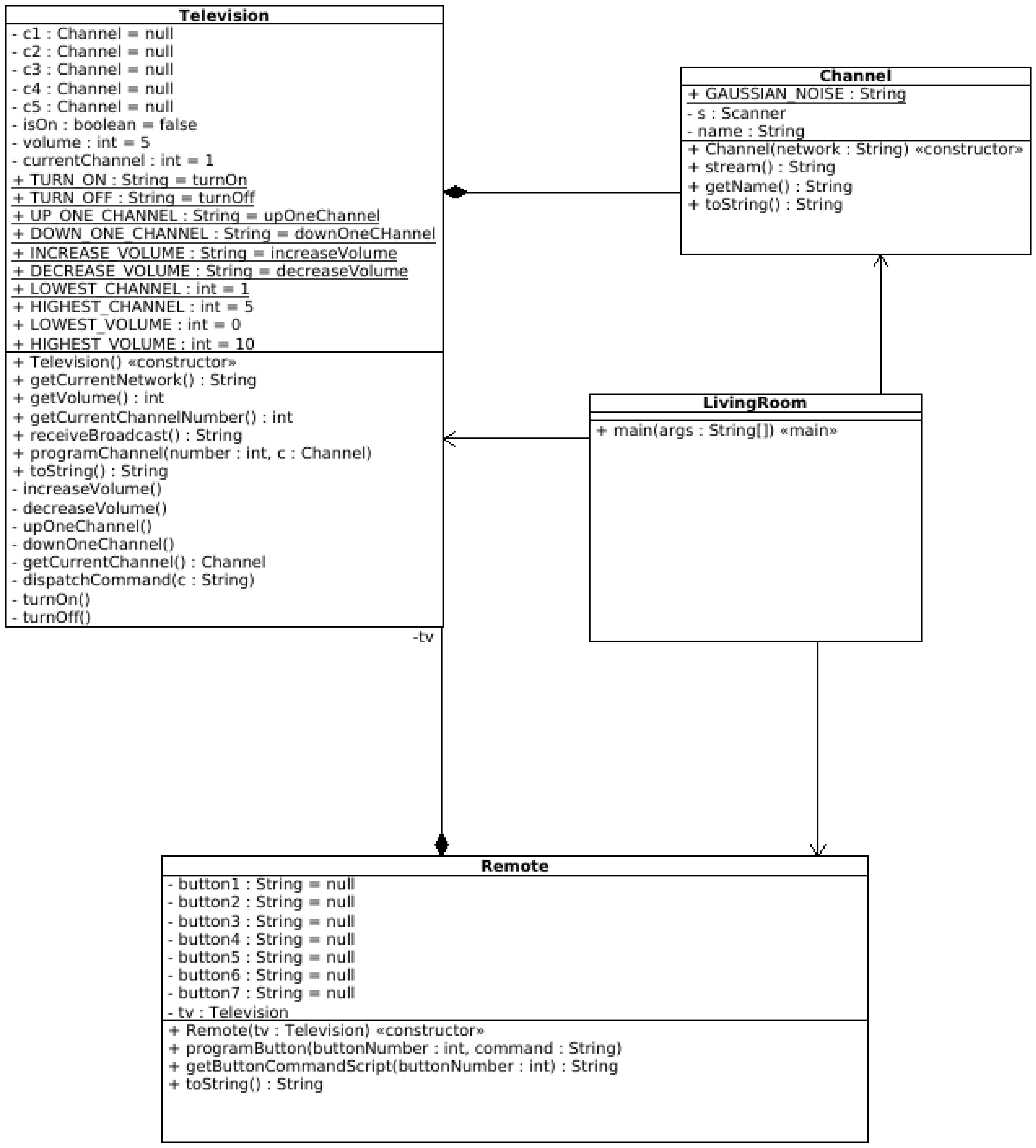
UML of Desired Program Spec

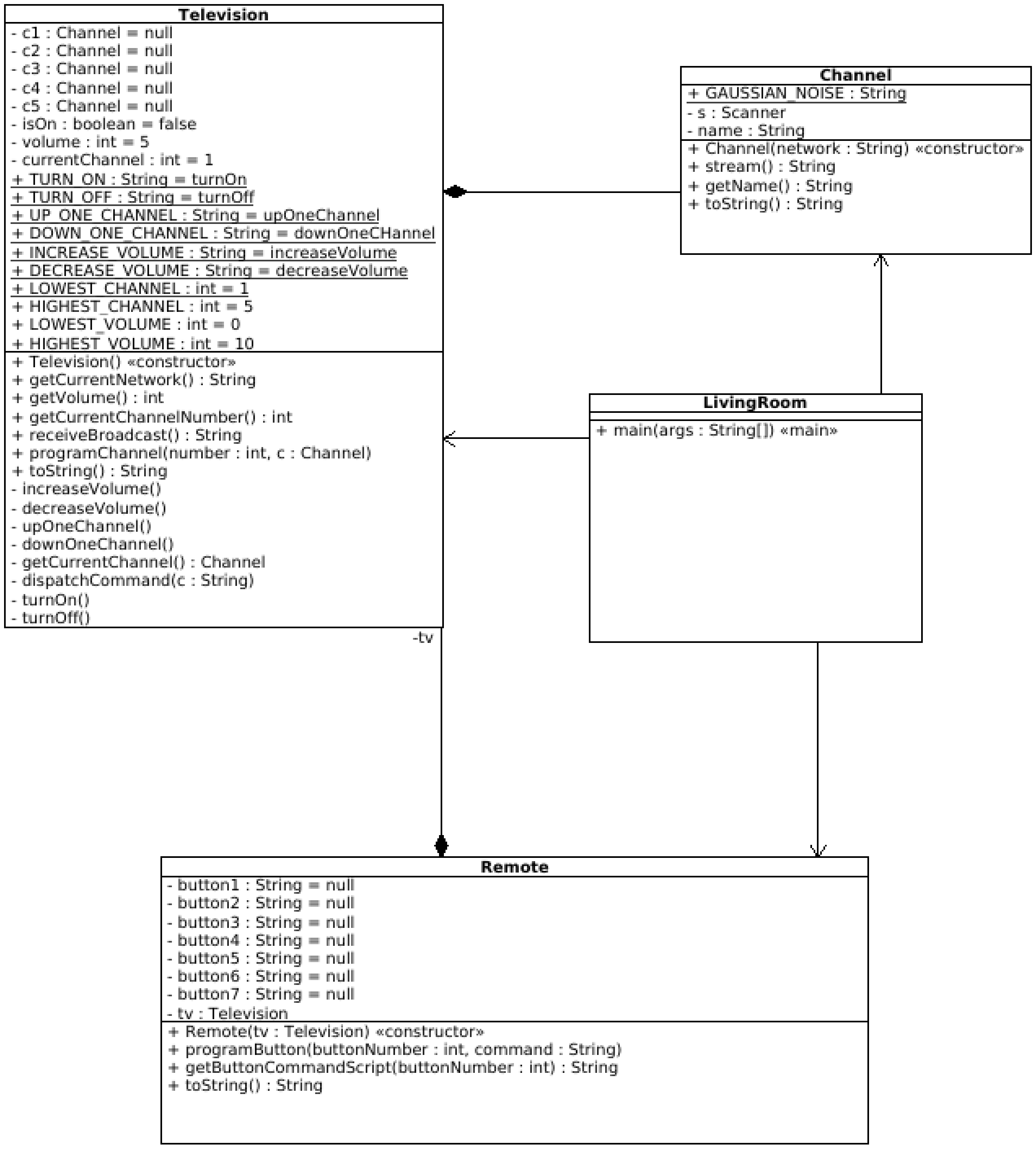

Customer
- wallet : double = 500
+ Customer() «constructor»
+ Customer(filename : String) «constructor»
+ spend(amount : double) : double
+ receive(amount : double)
+ checkWallet() : double
+ save(filename : String)
SlotMachine
- slot1 : char
- slot2 : char
- slot3 : char
- moneyPot : double
+ SlotMachine() «constructor»
+ SlotMachine(filename : String) «constructor»
+ pullLever(bet : double) : double
+ getMoneyPot() : double
+ save(filename : String)
+ toString() : String
GoodCasino
+ main(args : String []) «main»
+ play(c : Customer, s : SlotMachine, bet : double) : double

(BLANK FOR WRITING CODE, code for questions next pages)

Final Code

CSC110 Fall 2019


Television
- c1 : Channel = null
- c2 : Channel = null
- c3 : Channel = null
- c4 : Channel = null
- c5 : Channel = null
- isOn : boolean = false
- volume : int = 5
- currentChannel : int = 1
+ TURN_ON : String = turnOn
+ TURN_OFF : String = turnOff
+ UP_ONE_CHANNEL : String = upOneChannel
+ DOWN_ONE_CHANNEL : String = downOneCHannel
+ INCREASE_VOLUME : String = increaseVolume
+ DECREASE_VOLUME : String = decreaseVolume
+ LOWEST_CHANNEL : int = 1
+ HIGHEST_CHANNEL : int = 5
+ LOWEST_VOLUME : int = 0
+ HIGHEST_VOLUME : int = 10
+ Television() «constructor»
+ getCurrentNetwork() : String
+ getVolume() : int
+ getCurrentChannelNumber() : int
+ receiveBroadcast() : String
+ programChannel(number : int, c : Channel)
+ toString() : String
- increaseVolume()
- decreaseVolume()
- upOneChannel()
- downOneChannel()
- getCurrentChannel() : Channel
- dispatchCommand(c : String)
- turnOn()
- turnOff()
Channel
+ GAUSSIAN_NOISE : String
- s : Scanner
- name : String
+ Channel(network : String) «constructor»
+ stream() : String
+ getName() : String
+ toString() : String
LivingRoom
+ main(args : String[]) «main»
-tv
Remote
- button1 : String = null
- button2 : String = null
- button3 : String = null
- button4 : String = null
- button5 : String = null
- button6 : String = null
- button7 : String = null
- tv : Television
+ Remote(tv : Television) «constructor»
+ programButton(buttonNumber : int, command : String)
+ getButtonCommandScript(buttonNumber : int) : String
+ toString() : String


Television
- c1 : Channel = null
- c2 : Channel = null
- c3 : Channel = null
- c4 : Channel = null
- c5 : Channel = null
- isOn : boolean = false
- volume : int = 5
- currentChannel : int = 1
+ TURN_ON : String = turnOn
+ TURN_OFF : String = turnOff
+ UP_ONE_CHANNEL : String = upOneChannel
+ DOWN_ONE_CHANNEL : String = downOneCHannel
+ INCREASE_VOLUME : String = increaseVolume
+ DECREASE_VOLUME : String = decreaseVolume
+ LOWEST_CHANNEL : int = 1
+ HIGHEST_CHANNEL : int = 5
+ LOWEST_VOLUME : int = 0
+ HIGHEST_VOLUME : int = 10
+ Television() «constructor»
+ getCurrentNetwork() : String
+ getVolume() : int
+ getCurrentChannelNumber() : int
+ receiveBroadcast() : String
+ programChannel(number : int, c : Channel)
+ toString() : String
- increaseVolume()
- decreaseVolume()
- upOneChannel()
- downOneChannel()
- getCurrentChannel() : Channel
- dispatchCommand(c : String)
- turnOn()
- turnOff()
Channel
+ GAUSSIAN_NOISE : String
- s : Scanner
- name : String
+ Channel(network : String) «constructor»
+ stream() : String
+ getName() : String
+ toString() : String
LivingRoom
+ main(args : String[]) «main»
-tv
Remote
- button1 : String = null
- button2 : String = null
- button3 : String = null
- button4 : String = null
- button5 : String = null
- button6 : String = null
- button7 : String = null
- tv : Television
+ Remote(tv : Television) «constructor»
+ programButton(buttonNumber : int, command : String)
+ getButtonCommandScript(buttonNumber : int) : String
+ toString() : String

Class Remote

java.lang.Object
    Remote

```
public class Remote
extends java.lang.Object
```

This class represents a television remote for controlling a tv. It is designed to operate with the `Television` as controller that issues commands.

Constructor Summary

Constructors

Constructor	Description
`Remote(Television tv)`	Pairs the `Remote` object to a particular `Television`

Method Summary

All Methods   Instance Methods   Concrete Methods

Modifier and Type	Method	Description
`java.lang.String`	`getButtonCommandScript(int buttonNumber)`	
`void`	`press(int buttonNumber)`	Issues a command to the paired `Television`.
`void`	`programButton(int buttonNumber, java.lang.String command)`	Sets a particular button to contain a particular `String`.
`java.lang.String`	`toString()`	

Methods inherited from class java.lang.Object

`clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait`

Constructor Details

Remote

```
public Remote(Television tv)
```

Pairs the `Remote` object to a particular `Television`

Parameters:
`tv` - the television that the remote controls

Method Details

press

```
public void press(int buttonNumber)
```

Issues a command to the paired `Television`. What command is issued depends upon which button is pushed and what that button has been programmed to do.

Parameters:
`buttonNumber` - the number of the button that is to be pressed

programButton

```
public void programButton(int buttonNumber,
                          java.lang.String command)
```

Sets a particular button to contain a particular `String`. The `String` represents a sequence of commands to be issued to the `Television`

Parameters:
`buttonNumber` - the button to be programmed

`command` - represents a sequence of television commands that will happen when that button is pressed

getButtonCommandScript

```
public java.lang.String getButtonCommandScript(int buttonNumber)
```

Parameters:
`buttonNumber` - the button whose command script will be returned

Returns:
returns the command script for a particular button

toString

```
public java.lang.String toString()
```

Overrides:
`toString` in class `java.lang.Object`

Returns:
returns a catalog of the command script for each button of the remote

Class Channel

java.lang.Object
    Channel

```
public class Channel
extends java.lang.Object
```

Represents a channel that a television can tune to and receive a broadcast stream from. The broadcast stream or television station is represented by a text file that the channel consumes.

Field Summary

Fields

Modifier and Type	Field	Description
`static java.lang.String`	`GAUSSIAN_NOISE`	A constant representing the television fuzz that is returned before a channel is programmed onto a television

Constructor Summary

Constructors

Constructor	Description
`Channel(java.lang.String network)`	Creates a new `Channel` and gives it a network name.

Method Summary

All Methods Instance Methods Concrete Methods

Modifier and Type	Method	Description
`java.lang.String`	`getName()`	
`java.lang.String`	`stream()`	
`java.lang.String`	`toString()`	

Methods inherited from class java.lang.Object

`clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait`

Field Details

GAUSSIAN_NOISE

```
public static java.lang.String GAUSSIAN_NOISE
```

A constant representing the television fuzz that is returned before a channel is programmed onto a television

Constructor Details

Channel

```
public Channel(java.lang.String network)
        throws java.io.IOException
```

Creates a new `Channel` and gives it a network name. The network name is used to link to a file of the same name from which the channel will stream content.

Parameters:
`network` - the name of a tv station that will be used to find a file of the same name from which content will be delivered

Throws:
`java.io.IOException`

Method Details

stream

```
public java.lang.String stream()
```

Returns:
returns content from a broadcast network pulled from the network's file. If there is no content left, `GAUSSIAN_NOISE` is returned

getName

```
public java.lang.String getName()
```

Returns:
returns the name of the network the channel streams from

toString

```
public java.lang.String toString()
```

Overrides:
`toString` in class `java.lang.Object`

Returns:
also returns the name of the network the channel streams from

Class Television

java.lang.Object
    Television

```
public class Television
extends java.lang.Object
```

Represents an old-fashioned television that receives content live from an airwave broadcast

Field Summary

Fields

Modifier and Type	Field	Description
`static java.lang.String`	`DECREASE_VOLUME`	The commmand to decrease the volume one level
`static java.lang.String`	`DOWN_ONE_CHANNEL`	The command to move down one channel on the television
`static int`	`HIGHEST_CHANNEL`	The highest channel available on the tv
`static int`	`HIGHEST_VOLUME`	The highest volume the television can be set to
`static java.lang.String`	`INCREASE_VOLUME`	The command to increase the volume one level
`static int`	`LOWEST_CHANNEL`	The lowest channel available on the tv
`static int`	`LOWEST_VOLUME`	The lowest volume the television can be set to
`static java.lang.String`	`TURN_OFF`	The command to turn the television off
`static java.lang.String`	`TURN_ON`	The command to turn the television on
`static java.lang.String`	`UP_ONE_CHANNEL`	The command to move up one channel on the television

Constructor Summary

Constructors

Constructor	Description
`Television()`	Begins life off with its channel set to 1, its volume set to 5, and none of its channels programmed to a network

Method Summary

All Methods Instance Methods Concrete Methods

Modifier and Type	Method	Description
`int`	`getCurrentChannelNumber()`	
`java.lang.String`	`getCurrentNetwork()`	
`int`	`getVolume()`	
`void`	`programChannel(int number, Channel c)`	Sets one of the television's channels to a particular channel
`java.lang.String`	`receiveBroadcast()`	
`void`	`receiveCommand(java.lang.String c)`	Issues a sequence of commands to the television.
`java.lang.String`	`toString()`	

Methods inherited from class java.lang.Object

`clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait`

Field Details

TURN_ON

`public static final java.lang.String TURN_ON`

The command to turn the television on

See Also:
Constant Field Values

TURN_OFF

`public static final java.lang.String TURN_OFF`

The command to turn the television off

See Also:
Constant Field Values

UP_ONE_CHANNEL

`public static final java.lang.String UP_ONE_CHANNEL`

The command to move up one channel on the television

See Also:
Constant Field Values

DOWN_ONE_CHANNEL

`public static final java.lang.String DOWN_ONE_CHANNEL`

The command to move down one channel on the television

See Also:
Constant Field Values

INCREASE_VOLUME

`public static final java.lang.String INCREASE_VOLUME`

The command to increase the volume one level

See Also:
Constant Field Values

DECREASE_VOLUME

`public static final java.lang.String DECREASE_VOLUME`

The commmand to decrease the volume one level

See Also:

HIGHEST_CHANNEL

```
public static final int HIGHEST_CHANNEL
```

The highest channel available on the tv

See Also:
Constant Field Values

LOWEST_VOLUME

```
public static final int LOWEST_VOLUME
```

The lowest volume the television can be set to

See Also:
Constant Field Values

HIGHEST_VOLUME

```
public static final int HIGHEST_VOLUME
```

The highest volume the television can be set to

See Also:
Constant Field Values

Constructor Details

Television

```
public Television()
```

Begins life off with its channel set to 1, its volume set to 5, and none of its channels programmed to a network

Method Details

programChannel

```
public void programChannel(int number,
                           Channel c)
```

Sets one of the television's channels to a particular channel

Parameters:
`number` - the channel number to be programmed
`c` - the actual channel from which content will be streamed

receiveBroadcast

```
public java.lang.String receiveBroadcast()
```

receiveCommand

```
public void receiveCommand(java.lang.String c)
```

Issues a sequence of commands to the television. Each command is separated by a space and issued individually one at a time from left to right.

Parameters:
`c` - the sequence of commands the television will be asked to execute

getVolume

```
public int getVolume()
```

Returns:
returns the current volume level of the television

getCurrentChannelNumber

```
public int getCurrentChannelNumber()
```

Returns:
returns the number of the current channel

getCurrentNetwork

```
public java.lang.String getCurrentNetwork()
```

Returns:
returns the name of the current channel's network

toString

```
public java.lang.String toString()
```

Overrides:
`toString` in class `java.lang.Object`

Returns:
returns whether the set is on or off, information about the current channel, and the volume level

```
import java.util.Scanner;
import java.io.File;
import java.io.IOException;

public class Channel {
  public Channel(String network) throws IOException{
   this.name = network;
   s = new Scanner(new File(network + ".txt"));
  }

  public String stream(){
    String screen = GAUSSIAN_NOISE;

    if(s.hasNext()){
      screen = s.nextLine();
    }

    return screen;
  }

  public String getName(){
    return name;
  }

  public String toString(){
    return getName();
  }

  public static String GAUSSIAN_NOISE = "<GAUSSIAN STATIC>.34.2.92-
23-92-935.25029sjlkgset39690)(%*)(^*@#(@)(^)(@#%)TLj..c...gdkgkjp[]
[ppppfj4309506390346t#%*(@)@(%(@)//
fjkalsf;lsksjflkslskkdfjsk...<NO BROADCAST>";

  private Scanner s;
  private String name;
}
```

```
import java.io.IOException;

public class LivingRoom{
  public static void main(String [] args) throws IOException{
    Television tv = new Television();
    Remote remote = new Remote(tv);

    Channel nbc = new Channel("nbc");
    Channel mtv = new Channel("mtv");
    Channel abc = new Channel("abc");

    tv.programChannel(1, nbc);
    tv.programChannel(3, mtv);

    remote.programButton(1, Television.TURN_ON);
    remote.programButton(2, Television.TURN_OFF);
    remote.programButton(3, Television.UP_ONE_CHANNEL);
    remote.programButton(4, Television.DOWN_ONE_CHANNEL);
    remote.programButton(5, Television.INCREASE_VOLUME);
    remote.programButton(6, Television.DECREASE_VOLUME);


    String maxVolumeCommandScript = "";

    for(int i = 0; i < Television.HIGHEST_VOLUME; i++){
      maxVolumeCommandScript += (Television.INCREASE_VOLUME + " ");
    }

    remote.programButton(7, maxVolumeCommandScript);
    System.out.println(remote.getButtonCommandScript(7));

    String muteCommandScript = "";

    for(int i = 0; i < Television.HIGHEST_VOLUME; i++){
      muteCommandScript += (Television.DECREASE_VOLUME + " ");
    }

    remote.programButton(8, muteCommandScript);
    System.out.println(remote.getButtonCommandScript(8));

    System.out.println(tv);

    remote.press(1);
    remote.press(3);
    remote.press(3);
    remote.press(7);

    System.out.println(tv);

    System.out.println(tv.receiveBroadcast());
    System.out.println(tv.receiveBroadcast());

    remote.press(3);
    remote.press(3);
    remote.press(3);

    System.out.println(tv);

    System.out.println(tv.receiveBroadcast());
    System.out.println(tv.receiveBroadcast());

    remote.press(3);
    System.out.println(tv);
    System.out.println(tv.receiveBroadcast());
  }
}
```

```
public class Remote{
  public Remote(Television tv){
    this.tv = tv;
  }

  public void press(int buttonNumber){
    switch(buttonNumber){
      case 1:
        tv.receiveCommand(button1);
          break;
      case 2:
          tv.receiveCommand(button2);
          break;
      case 3:
          tv.receiveCommand(button3);
          break;
      case 4:
          tv.receiveCommand(button4);
          break;
      case 5:
          tv.receiveCommand(button5);
          break;
      case 6:
          tv.receiveCommand(button6);
          break;
      case 7:
          tv.receiveCommand(button7);
          break;
      case 8:
          tv.receiveCommand(button8);
          break;
    }
  }

  public void programButton(int buttonNumber, String command){
    switch(buttonNumber){
      case 1:
        button1 = command;
          break;
      case 2:
          button2 = command;
          break;
      case 3:
          button3 = command;
          break;
      case 4:
          button4 = command;
          break;
      case 5:
          button5 = command;
          break;
      case 6:
          button6 = command;
          break;
      case 7:
          button7 = command;
          break;
      case 8:
          button8 = command;
          break;
    }
  }

  public String getButtonCommandScript(int buttonNumber){
      String commandScript = "";

      switch(buttonNumber){
        case 1:
          commandScript = button1;
          break;
        case 2:
          commandScript = button2;
          break;
        case 3:
          commandScript = button3;
          break;
        case 4:
          commandScript = button4;
          break;
        case 5:
          commandScript = button5;
          break;
        case 6:
          commandScript = button6;
          break;
        case 7:
          commandScript = button7;
          break;
        case 8:
          commandScript = button8;
          break;
      }

      return commandScript;
  }


  public String toString(){
    String strRemote = "";

    strRemote += ("button1: " + button1);
    strRemote += ("button2: " + button2);
    strRemote += ("button3: " + button3);
    strRemote += ("button4: " + button4);
    strRemote += ("button5: " + button5);
    strRemote += ("button6: " + button6);
    strRemote += ("button7: " + button7);
    strRemote += ("button8: " + button8);

    return strRemote;
  }

  private String button1;
  private String button2;
  private String button3;
  private String button4;
  private String button5;
  private String button6;
  private String button7;
  private String button8;

  private Television tv;
}
```

```
public class **Television**{
  public Television(){
    isOn = false;
    currentChannel = 1;
    volume = 5;
  }

  public void programChannel(int number, Channel c){
    switch(number){
      case 1:
        c1 = c;
        break;
      case 2:
          c2 = c;
          break;
      case 3:
          c3 = c;
        break;
      case 4:
          c4 = c;
          break;
      case 5:
          c5 = c;
          break;
      default:
          c5 = c;
    }
  }

  private void increaseVolume(){
   if(volume < HIGHEST_VOLUME){
     volume++;
   }
  }

  private void decreaseVolume(){
    if(volume > LOWEST_VOLUME){
      volume--;
    }
  }

  private void upOneChannel(){
    if(currentChannel < HIGHEST_CHANNEL){
      currentChannel++;
    } else {
      currentChannel = LOWEST_CHANNEL;
    }
  }

  private void downOneChannel(){
    if(currentChannel > LOWEST_CHANNEL){
      currentChannel--;
    } else {
      currentChannel = HIGHEST_CHANNEL;
    }
  }

  public String receiveBroadcast(){
    String stream = "<Tv is off>";
    if(isOn){
      if(getCurrentChannel() != null){
        stream = getCurrentChannel().stream();
      } else{
        stream = Channel.GAUSSIAN_NOISE;
      }
    }
    return stream;
  }

  private Channel getCurrentChannel(){
    Channel channelToReturn = null;

    switch(currentChannel){
      case 1:
        channelToReturn = c1;
          break;
      case 2:
          channelToReturn = c2;
          break;
      case 3:
          channelToReturn = c3;
          break;
      case 4:
          channelToReturn = c4;
          break;
      case 5:
          channelToReturn = c5;
          break;
    }

    return channelToReturn;
  }

  private void dispatchCommand(String c){
    switch(c){
      case TURN_ON:
          turnOn();
          break;
      case TURN_OFF:
          turnOff();
          break;
      case UP_ONE_CHANNEL:
          upOneChannel();
          break;
      case DOWN_ONE_CHANNEL:
          downOneChannel();
          break;
      case INCREASE_VOLUME:
          increaseVolume();
          break;
      case DECREASE_VOLUME:
          decreaseVolume();
          break;
    }
  }

  private void turnOn(){
    isOn = true;
  }

  private void turnOff(){
    isOn = false;
  }

  public void receiveCommand(String c){
    if(isOn){
      String [] splitted = c.split(" ");

      for(int i = 0; i < splitted.length; i++){
        dispatchCommand(splitted[i]);
      }
    } else{
     if(c.equals("turnOn")){
       dispatchCommand(c);
     }
    }
  }

  public int getVolume(){
    return volume;
  }

  public int getCurrentChannelNumber(){
    return currentChannel;
  }

  public String getCurrentNetwork(){
    return getCurrentChannel().getName();
  }

  public String toString(){
    String str = "";

    str += ("\nSet: " + (isOn ? "On" : "Off") + "\n");
    str += ("Channel " + getCurrentChannelNumber() + ": " +
getCurrentChannel() + "\n");
    str += ("Volume: " + getVolume() + "\n\n");

    return str;
  }

  public static final String TURN_ON = "turnOn";
  public static final String TURN_OFF = "turnOff";
  public static final String UP_ONE_CHANNEL = "upOneChannel";
  public static final String DOWN_ONE_CHANNEL = "downOneChannel";
  public static final String INCREASE_VOLUME = "increaseVolume";
  public static final String DECREASE_VOLUME = "decreaseVolume";

  public static final int LOWEST_CHANNEL = 1;
  public static final int HIGHEST_CHANNEL = 5;
  public static final int LOWEST_VOLUME = 0;
  public static final int HIGHEST_VOLUME = 10;

  private Channel c1;
  private Channel c2;
  private Channel c3;
  private Channel c4;
  private Channel c5;

  private boolean isOn;
  private int volume;
  private int currentChannel;

}
```

Files

//lines in < > are not part of the file
//each bracket group < > < > is a separate file

```
<!--begin vh1.txt-->

vh1 signal 1
vh1 signal 2
vh1 signal 3
vh1 signal 4
vh1 signal 5
vh1 signal 6
vh1 signal 7
vh1 signal 8
vh1 signal 9
vh1 signal 10

<!--end of file vh1.txt -->tbs signal 1


<!--begin tbs.txt -->

tbs signal 2
tbs signal 3
tbs signal 4
tbs signal 5
tbs signal 6
tbs signal 7
tbs signal 8
tbs signal 9
tbs signal 10

<!--end of file tbs.txt →


<!--begin qvc.txt →

qvc signal 1
qvc signal 2
qvc signal 3
qvc signal 4
qvc signal 5
qvc signal 6
qvc signal 7
qvc signal 8
qvc signal 9
qvc signal 10

<!--end of file qvc.txt →


<!begin abc.txt-→

abc signal 1
abc signal 2
abc signal 3
abc signal 4
abc signal 5
abc signal 6
abc signal 7
abc signal 8
abc signal 9
abc signal 10

<!--end of abc.txt-→
```

```
<!--begin cbs.txt-→

cbs signal 1
cbs signal 2
cbs signal 3
cbs signal 4
cbs signal 5
cbs signal 6
cbs signal 7
cbs signal 8
cbs signal 9
cbs signal 10

<!--end of cbs.txt →


<!--begin cc.txt →

cc signal 1
cc signal 2
cc signal 3
cc signal 4
cc signal 5
cc signal 6
cc signal 7
cc signal 8
cc signal 9
cc signal 10

<!--end of cc.txt —→


<!-- cmt.txt -->

cmt signal 1
cmt signal 2
cmt signal 3
cmt signal 4
cmt signal 5
cmt signal 6
cmt signal 7
cmt signal 8
cmt signal 9
cmt signal 10
cnn signal 1
cnn signal 2
cnn signal 3
cnn signal 4
cnn signal 5
cnn signal 6
cnn signal 7
cnn signal 8
cnn signal 9
cnn signal 10

<!--end of cnn.txt-->
```

```
<!--begin cn.txt →

cn signal 1
cn signal 2
cn signal 3
cn signal 4
cn signal 5
cn signal 6
cn signal 7
cn signal 8
cn signal 9
cn signal 10


<!--end of cn.txt-→


<!--begin hbo.txt —->

hbo signal 1
hbo signal 2
hbo signal 3
hbo signal 4
hbo signal 5
hbo signal 6
hbo signal 7
hbo signal 8
hbo signal 9
hbo signal 10

<!--end of hbo.txt-->


<!--begin mtv.txt-->

mtv signal 1
mtv signal 2
mtv signal 3
mtv signal 4
mtv signal 5
mtv signal 6
mtv signal 7
mtv signal 8
mtv signal 9
mtv signal 10

<!--end file mtv.txt-->


<!--begin nbc.txt →

nbc signal 1
nbc signal 2
nbc signal 3
nbc signal 4
nbc signal 5
nbc signal 6
nbc signal 7
nbc signal 8
nbc signal 9
nbc signal 10

<!--end of file nbc.txt-->
```