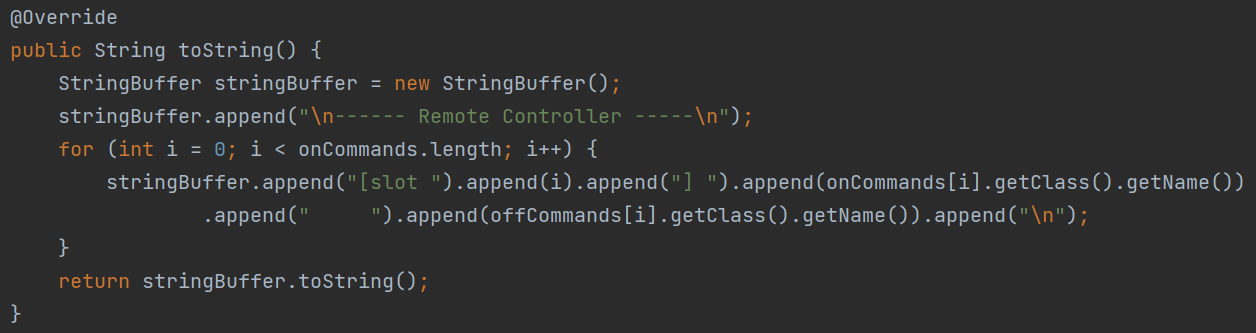
**Command:**

* Open Demos>Java>command-pattern in IntelliJ and navigate to src>main>java
* Walk the students through the methods in Light
* Create an interface called Command with an abstract method called execute
* Create a class called LightOnCommand that implements Command
  + Create a private final field for a Light
  + Generate a constructor that takes and assigns a Light
  + Override the execute method to call light.on();
* Create a class called SimpleRemoteControl
  + Create a private final field for a Command called slot
  + Create a setter for slot
  + Create a method called buttonWasPessed that calls execute on slot
* In Main, create a SimpleRemoteControl, a Light, and a LightOnCommand
* Call setCommand on the remote and pass in the instance of LightOnCommand
* Call buttonWasPressed on the remote and run the program
* Create a new class called RemoteControl – this will represent a remote controller with 3 sets of on/off buttons that can be programmed to many different electronics in your home
  + This class will contain 2 arrays of Commands, one called onCommands and offCommands
  + Create a constructor that takes no args, sets the Command arrays to arrays with 3 elements, create an instance of noCommand and use a for loop to set all commands in the arrays to the instance of noCommand
* Create a new class called NoCommand that implements Command and overrides execute with no body
* In RemoteControl, create a setCommand method that takes a slot number, an onCommand, and an offCommand, and sets the commands at that slot number to the on and off commands passed in
  + Create a method called onButtonWasPushed that takes an int for the slot number and calls execute on the onCommand at that slot
  + Create an offButtonWasPushed that mirror the onButtonWasPushed method
  + Override toString() with the following code:



* Create a class called LightOffCommand that is the same as the LightOnCommand class except execute() calls light.off()
* Clear all of the code out of Main
* Create 2 instances of Light, one in the living room and one in the kitchen
* Create the on and off commands necessary for each instance
* Set the remote commands to the command instances
* Print remoteControl
* Call onButtonWasPushed and offButtonWasPushed for each slot on the remote – run the program
* Add an abstract undo method to the Command interface
  + Override it in every concrete command class to call the opposite command (ie. LightOnCommand -> light.off())
  + The body of undo() should be empty in NoCommand
  + Add a previousCommand field to RemoteControl
  + In the constructor, set previousCommand to noCommand
  + In onButtonWaPushed and offButtonWasPushed, set previousCommand to the slot being executed
  + Add an undoButtonWasPushed method that calls previousCommand.undo()
  + In toString, add the following code before the return:
    - stringBuffer.append("[undo] ").append(this.previousCommand.getClass().getName()).append(" \n");
* In Main, comment out the calls to onButtonWasPushed and offButtonWasPushed, except for the first
* After the call to onButtonWasPushed(0) print the remoteControl again
* Call undoButtonWasPushed then print the remoteControl again – run the program

**Template Method:**

* Open Demos>Design-Patterns>template-method in IntelliJ and navigate to src>main>java
* Walk students through the Creature, Orc, and Elf classes as the character classes available in a game
* Create an abstract class called GameAI
  + Create 3 private ArrayLists for holding structures, resources, and units. Units are Creatures and the others are Strings
  + Create a public method called takeTurn that takes a GameAI and returns void
  + In takeTurn, call collectResources, buildStructures, buildUnits, and attack, which takes the GameAI argument
  + Create abstract methods for collectResources, buildStructures, and buildUnits that all return void
  + Create the attack method that take an enemy GameAI, checks that it has units, attacks the enemies head unit (units.get(0)) with its own, and removes a unit if killed
  + Create getters for all the ArrayLists
* Create an OrcsAI that extends GameAI and overrides all of its abstract methods
  + BuildStructures should add “Straw Hut” to the structures and pint the number of Straw Huts
  + buildUnits should add an Orc to the units and print the number of units
  + collectResources should add 2 Bone and print the number of bones
* Create an ElvesAI that extends GameAI and overrides all of its abstract methods
  + BuildStructures should add “Tree House” to the structures and pint the number of Tree Houses
  + buildUnits should add an Elf to the units and print the number of units
  + collectResources should add a Flower and print the number of flowers
* In Main, create an OrcsAI and an ElvesAI and have them take a few turns back and forth – run the program

**Strategy:**

* Open Demos>Design-Patterns>strategy-pattern in IntelliJ and navigate to src>main>java
* Walk students through what CustomerBill is doing (holding order items, adding to order, getting total, printing receipt)
* Create a BillingStrategy interface with the abstract method getSubtotal that takes a HashMap<String, Double> and returns a double
* Create a PickUpOrderBillingStrategy class that implement BillingStrategy
  + Override getSubtotal to have the same body as the CustomerBill subtotal method
* In CustomerBill, add a field for a BillingStrategy and generate a constructor that takes and sets the strategy
  + Update getSubtotal to call the strategy’s getSubtotal method
* In Main, create a new CustomerBill and pass in the PickUpOrderBillingStrategy, add a few items to it, then print the receipt – run the program
* Create a new DeliveryOrderBillingStrategy that accounts for a delivery fee when calculating the subtotal
* In CustomerBill, add a new method checkDeliveryFee, that checks if the strategy is an instance of DeliveryOrderBillingStrategy and prints a line for the delivery fee
  + Call this method in printReceipt, just before the subtotal line
* In Main, switch the CustomerBill strategy to DeliveryOrderBillingStrategy – run the program
* Create a new DeliveryOrderWithSurgeBillingStrategy that accounts for a surge pricing delivery fee when calculating the subtotal
* In CustomerBill, update checkDeliveryFee to check if the strategy is an instance of DeliveryOrderWithSurgeBillingStrategy and prints a line for the surge delivery fee
* In Main, switch the CustomerBill strategy to DeliveryOrderWithSurgeBillingStrategy – run the program

**Observer:**

* Open Demos>Design-Patterns>observer-pattern in IntelliJ and navigate to src>main>java
* Create a class called NewsAgency that has a String field for news and an ArrayList of NewsOutlets
* Create an abstract class called NewsOutlet that has String fields for news and name
  + Generate a constructor that only takes and sets the name
  + Create abstract methods called update and getNews that both return void
* In NewsAgency, create a method called addObserver that takes a NewsOutlet and adds it to the ArrayList of outlets
  + Create a method called removeObserver that takes a NewsOutlet and removes it from the ArrayList of outlets
  + Create a method called setNews that takes a String and returns void, in the body, set the objects news to the passed in value and loop through the outlets and call their update method with the passed in value
* Create a NewsChannel class that extends NewsOutlet
  + Generate a constructor that calls the super constructor
  + Override the update method to set this.news to the passed in value
  + Override getNews to print the channel’s name and the news as if spoken by an anchor
* Create a NewsWebsite class that extends NewsOutlet
  + Generate a constructor that calls the super constructor
  + Override the update method to set this.news to the passed in value
  + Override getNews to print the website’s name and the news as if it was a recent blog post
* In Main, create instances of NewsAgency, NewsChannel, and NewsWebsite
* Call addObserver on the news agency twice with each outlet as an arg
* Call setNews on the agency and pass in some news
* Call getNews on each outlet – run the program