**Classes & Objects:**

* Open demos>Java>classes-and-objects in IntelliJ
* Navigate to src>main>java and create a new class called Student
* Give the Student class 3 attributes, ex. name, agileTeam, gradePointAverage
  + Make name and agileTeam final as they will not change
  + Make gradePointAverage private so it cannot be accessed outside of student
* Create a constructor for Student that takes the name and agileTeam but sets gradePointAverage to 0 by default
* Generate a getter and setter for gradePointAverage since it’s private and can’t be accessed outside the class
* Open Main and create a Student inside of the main method
* Print the student’s name and agile team
* Try to print the student’s gradePointAverage
  + This will cause an error b/c the gradePointAverage is private
  + Switch the direct access of gradePointAverage to the getter method and run the program again
* Use the gradePointAverage setter method to set the gradePointAverage to 96.8 and run the program again
* The students gradePointAverage now prints 98.6

**Non-access Modifiers:**

* Open Demos>Java>non-access-modifiers in IntelliJ and navigate to src>main>java
* Create a new example class
* Within the example class create a final variable (ex. final String name) and a final getter method for it
* Try to reassign the final variable, this will cause an error because final variables cannot be reassigned
* Create a static variable called count and assign it to 0
* Create a static method that resets the count to 0;
* Create a constructor that increments count
* Create an abstract method called print count and make the example class abstract
* Create a child class to extend the example class
* Override the printCount method and have it print the count
* Attempt to override getName – this will cause an error because getName is final
* In Main, create an instance of the child class and call printCount on it – run the program
* Create 2 more instances of the child class before the printCount call – run the program
* The count is now 3 no matter with instance you call printCount on b/c they all share the same count
* Attempt to call resetCount on one of the instances of the child class – this will not work because resetCount it static and should be called on the class itself
* Correct the call to resetCount, make another call to printCount and run the program

**Inheritance:**

* Open Demos>Java>inheritance in IntelliJ
* Navigate to src>main>java and create a new class called Employee
  + Give the employee class the attributes employeeID, name, email, and salary
  + Create a constructor that takes all attributes except id, which is set by Math.random()
  + Create getters for all attributes, and a setter for salary
  + Create employee methods onBoard, offBoard, and grabCoffee and make them print that action completed (i.e. “employee was onboarded”)
* Create a new class called Developer and make it extend employee
  + Get rid of the error by creating a constructor that calls super()
  + Give Developer the attribute currentProject and set it to “on the bench” in the constructor
  + Create a getter and setter for currentProject
  + Add the method code() that prints that the developer is coding
* Open Main and create and Employee and a Developer
  + Call onBoard and grabCoffee on employee and developer
  + Call code on developer – show that code cannot be called on employee
  + Call offBoard on both and run the program

**Composition:**

* Open Demos>Java>composition in IntelliJ
* Naviagate to src>main>java and create a class called Room
* Give Room the attributes squareFootage and wallColor
* Generate a constructor and getters and setters (no setter for squareFootage)
* Create a class called Building
* Give it an ArrayList of Rooms
* Generate a constructor, getter, and setter
* Create a method that loops through the rooms in the ArrayList and returns the sum of their square footage
* In Main, create a few rooms and add them to an arrayList, create a building and pass in the arrayList of rooms
* Print the building’s total square footage

**Abstract Classes:**

* Open Demos>Java>abstract-classes in IntelliJ and navigate to src>main>java
* Create an abstract class called BoardGame
  + Give it the following attributes: boardHeightInInches, boardWidthInInches, and a list of gamePieces
  + Generate a constructor with all of the above attributes
  + Generate getters for all attributes and a setter of gamePieces
  + Create a placeGameBoard method that prints "The getBoardHeightInInches() inch by getBoardWidthInInches() inch game board has been placed on the table.“
  + Declare the following abstract classes: setup and play
* Create a Checkers class that extends BoardGame
  + Create a constructor that takes no arguments and calls super to set the board height and width to 8 and pass an empty list of game pieces
  + Override setup and play
  + Create a private method called createDiscs that creates and empty arrayList and loops 12 times to add red discs to the list and another 12 to add black discs
  + In setup, call setGamePieces and pass in the createDiscs method, then print “Red and Black discs have been placed on the board“
  + In play, print "2 players are playing Checkers!“
* Create a Sorry class that extends BoardGame
  + Create a constructor that takes no arguments and calls super to set the board height and width to 10 and pass an empty list of game pieces
  + Override setup and play
  + Create a private method called createPawns that creates and empty arrayList and loops 4 times each to add red pawns, green pawns, yellow pawns, and blue pawns
  + In setup, call setGamePieces and pass in the createPawns method, then print "Players have picked their pawn colors and placed all 4 in the designated start space.“ and "The deck has been shuffled and placed in the designated spot."
  + In play, print "2-4 players are playing Sorry!“
* In Main, create a new checkers game and a new sorry game and call the follow methods on each: placeGameBoard, setup, and play
* Run the program

**Method Overloading:**

* Open Demos>Java>method-overloading in IntelliJ
* Navigate to src/main/java/Main.js
* Create a method called printUserData that takes a user and souts their name and age
* Show the students that a user class has already been created that has the attributes name and age, which keeps us from getting an error when writing this method
* Overload the method by creating another version that takes a String for name and an int for age and souts the same sentence
* In the main method, create a User then call both versions of the method
* Try to create another version of the method that takes a User and returns a String
  + This will cause an error because printUserData(User) has already been defined

**Method Overriding:**

* Open Demos>Java>inheritance in IntelliJ
* Open the Employee you made earlier
* Change the onboarding method to a list of steps the employee will complete while onboarding ex.:
  + sout(“this.name setup their laptop”);
  + sout(“this.name checked their email”);
  + sout(“this.name completed degreed employee training”);
* Open Developer and override the onBoard method to include the superclass’s code and add a step for completing developer training
* Open Main and call onBoard on employee and developer