# Lessons

* <https://www.youtube.com/playlist?list=PL_c9BZzLwBRKIMP_xNTJxi9lIgQhE51rF>
* <https://www.w3schools.com/java/default.asp>

# 100. Conclusion!!!

* Generic Programming
* Web Development with Java
* User Interface development
* Web Applications
* Desktop Applications
* Android Applications
* Topics to Java
  + DataBase Reading
    - Input and Output files
  + Error Handling/debugging
  + HashTables
  + GUI <https://www.youtube.com/playlist?list=PL6gx4Cwl9DGBzfXLWLSYVy8EbTdpGbUIGhttps://www.youtube.com/playlist?list=PL6gx4Cwl9DGBzfXLWLSYVy8EbTdpGbUIG>

# 98. Intro to enum

* Enum allows you to have a list of possible values for a variable
* Example in User class

| public enum status {  active,  inactive,  probation};  public status status; |
| --- |

* Example in operating class

| Student s = new Student();  S.status = s.status.probation;  sysout(s.status); //outputs probation |
| --- |

# 97. Final Classes

* A final class prevents anything from extending it.
* If class Teacher is final, then public class Teach extends Teacher will give you an error

# 96. Final Methods

* If a method in a parent class is described as final the subclasses can’t override that method

# 95. Creating an interface

* To have an interface requires to have the method in the class
* Steps in eclipse
  + Right-click the Project -> New -> Interface
* Example creating a new interface

| Public interface Talk {  void sayHello();  } |
| --- |

* Example in student class

| Public class Student extends User implements Talk{  Public void sayHello(){}  } |
| --- |

# 94. Intro to Interfaces

* Interfaces define behavior, and how you work with something



# 92. Invoke Parent Class Methods with Super

* You use the super method in your child's class if you want to access the parent class version of a method in the parent class

# 91. Custom Constructor

* Example in Student class

| Public Student(String fn, String ln){  firstName = fn;  lastName = ln;  } |
| --- |

* Example in operating class

| Student s = new Student(“Dakota”, “Gullicksen”); |
| --- |

# 90. Creating the Default Constructor

* Example in Student class

| Public Student() {  sysout(“Creating a Student”);  } |
| --- |

* Example in operating class

| Student s = new Student(); //Will output the sysout when it is “hit” |
| --- |

# 89. Intro to Constructors

* Like a method but when you create a constructor there’s no return type and the way to invoke it is different. Its a way that at the very beginning the user has valid values

| User u = new User(); = public User(){} //for User class  User u = new User(“Dakota”); = public User(string n){firstName = n;} //firstName is already stated as a string data type  Implicit Default Constructor new User(){}  Custom constructor User(‘Dakota”){} |
| --- |

# 86. Abstract Method

* Public abstract void sayHello();

# 85. Abstract Class

* Public abstract class User {} //abstract prevents us from instantiating a user
* So if we say User u = new User(); in the operating class it’s going to give us an answer. The only way we can get a value from the user class is if we extend it in another class. You still get the benefits.

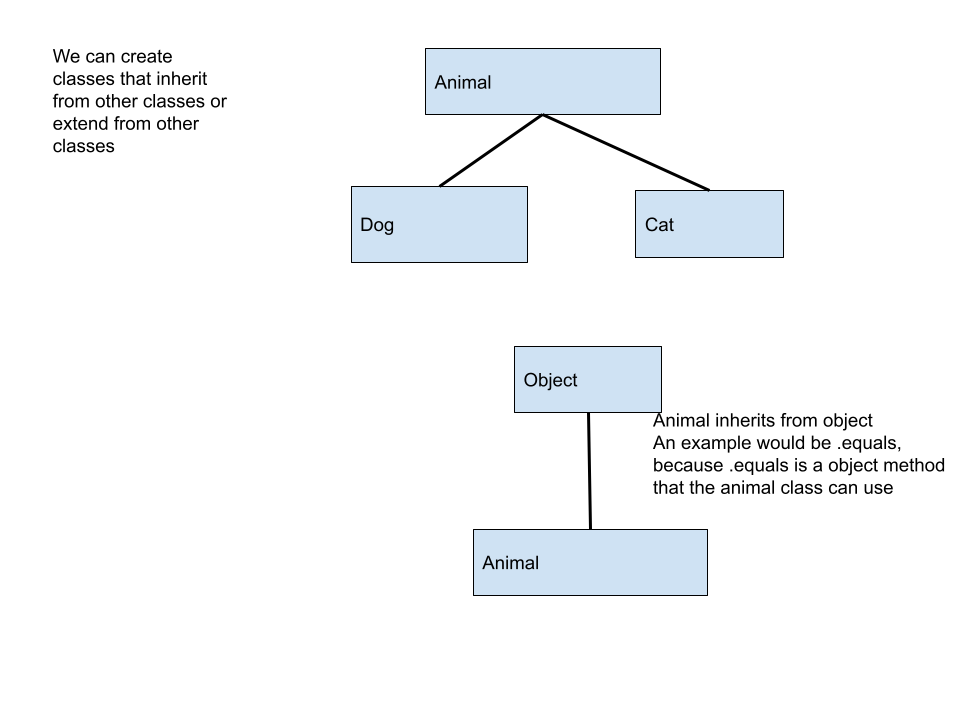
# 83. Virtual in Java

* hello() from the base class. hello() can be overridden in the subclass. If this is used in a subclass the base class method needs to be labeled virtual, but it's done implicitly. Most methods are going to be labeled virtual by default. Ones that aren’t virtual are static methods, the same as private.

# 82. Working with Inheritance

| Public class Student extends User {} //User is a parent class the Student is the subclass |
| --- |

# 81. Intro to Inheritance



# 80. Passing by Value or Reference

* Java is pass by value meaning if we pass some argument to a method it passes that value to the parameter. If we are working with an object the value passed in is the location of memory.

# 79. Returning Custom Objects

* Example in User Class

| Public static User findUser(List<User> users, User u){  for(User user : users){  if(user.equals(u)){  Return user;  }  }  Return null;  } |
| --- |

* Example in operating Class

| User search = new User();  search.setFirstName(“Dakota”);  search.setLastName(“Gullicksen”);  User found = User.findUser(users, search);  sysout(found); //outputs User [getFullName( ) = Dakota Gullicksen], null if the name isn’t found |
| --- |

# 78. Overload the Search to Take in a User Object

* Example in User class

| Public static int searchList (List<User> users, User u){  Return searchList(users, u.getFullName());  } |
| --- |

* Example in operating class

| sysout(User.searchList(users, you)); //outputs 1 because of index 1 |
| --- |

# 77. Override Equals

* hashcode()?
* equals()?

# 76. Override toString

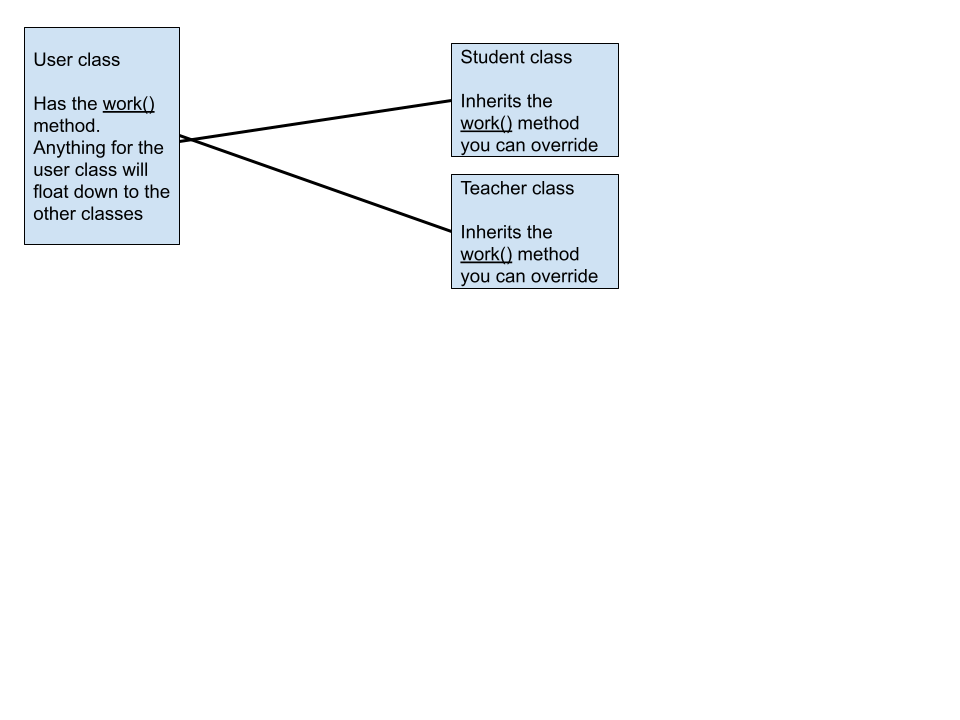
* Example in User class

| Public String toString() {  Return “User [getFullName( )=” + getFullName( ) + “]”;  } |
| --- |

* Example in operating class

| sysout(me); //prints User [getFullName( )=Dakota Gullicksen “] if Dakota Gullicksen is pre-defined |
| --- |

# 75. Method Overriding



# 74. Searching a List for Custom Objects

* Example

| Public static void searchList (List<User> users, String fn. String ln){  Return searchList(users, fn + “ “ + ln); //goes to the seachList overload  method with the list and names  }  Public static int searchList(List<User> users, String fullName){  For (int i = 0; i < users.size(); i++){ //searches for every i index of the list  if(users.get(i).getFullName() .equals(fullName)){  Return i; //returns the index number  }  }  Return -1; //if the person doesn’t exist  } |
| --- |

# 72. Intro to Method Overloading and Optional Parameters

| Public void test (int x)  Public void test (string x)  Public void text (int x, int y)  variable.test(5); //calls on the first  variable.test(“5”); //calls on the second  variable.test(5, 4); //calls on the third |
| --- |

# 71. Method to take an ArrayList of Custom Type

* Example

| public static void printUser(List<User> users){  for(User u : users){  sysout(u.getFullName());  }  } |
| --- |

# 70. Creating a Static Method

* In the custom class you would have either method as
  + public void method(){} //Instance
  + public static void method(pram){} //static, generally related to users but not needed as an instance to users
* Then call it in your overall program as
  + CustomClass variable = new CustomClass();
  + CustomClass.method(variable);

# 68. Taking Custom Types as Arguments

* Example

| Public void printUser(User u){  sysout(u.getFullName);  }//You can then use this in main by calling on it by assigning your main class you’re working on with a variable name. Like  MySweetProgram m = new MySweerProgram();  m.printUser(Users.get(0)); |
| --- |

# 67. Creating Custom Type in Loop

* Example

| String[ ] firstNames = {“Dakota”, “Walker”, “Haley”};  String[ ]lastNames = {“Gullicksen”, “Gullicksen”, “Price”};  List<User> users = new Arraylist<User>();  for(int i = 0; i < firstNames.length; i++){  User u = new user();  u.setFirstName(firstNames[i]);  u.setLastName(firstNames[i]);  users.add(u);  }  for(User u : users){  sysout(u.getFullName);  } |
| --- |

# 66. ArrayList of Custom Type

* List<CustomClass> arrayList= new Arraylist<CustomClass>();
* arrayList.add(variable);
* Example

| List<User> users = new Arraylist<User>();  users.add(user); |
| --- |

# 63. Create a Getter and 64. Create a Setter

* provide public get and set methods to access and update the value of a private variable

| public void setVariable(dataType variable){  this.variable = variable;  }  public String getVariable(){  return variable;  } |
| --- |

# 62. Encapsulation

* A getter gets us the value from a variable that is either public to every class or private in the class
* A setter allows us to update the value, making the variable have a new value

# 61. Return Statement

* Return - what the method returns within the data type is set for the method
  + Return; //returns nothing
  + Return “string”; //returns a string if the data type is string
  + Return #; //returns a number if the return type is int or double

# 60. Arguments and Parameters in Methods

* Argument the values that you pass inside a method
* Parameters act as variables inside the method

# 59. Creating Methods

* Within the class curly brackets you would put your method. If you want the method to be accessed by other classes make it public else make it private.

| public class Class {  public void method();  private void method2();  } |
| --- |

# 56. Methods

* Public or private, return data type, and method name are inside of the class.
* Example

| public class Person {  public String name;  public String talk(){  return “Hi! I’m “ + name;  }  }    Public static void main(){  Person me = new Person(); //constructor  me.name = “Dakota”;  me.talk();  } |
| --- |

# 55. Public vs Private

* Public - the most open access level that can be used outside of the class
* Private - a variable that is only used inside of a class, you can only access it through methods or accessed inside class definition
* Lets say there’s a private field called name, name has 2 methods setName and getName that can only access the name in the class
  + Setter - changes and sets the name
  + Getter - allows us to get the information

# 54. Fields

* Field- variable inside a class
* Example

| Person is a class with name field  {Public string name;}  main{  Person me = new Person();  me.name = “Dakota”;} |
| --- |

# 52. Intro to Object Oriented Programming (OOP) and 53. Class vs Object

* Objects - are the specific thing from the class
* Allows us to create large programs so we don’t have to code everything by hand
* Class - a combination of variables and methods to represent a general layout of things. Basically creating a custom data type.
  + You instantiate the class to create objects.
  + The class will be defined inside a file
  + The objects will be defined as a variable
* Example

| public class Person { }  Person x //where Person is the class and x is the variable |
| --- |

# 51. Sort and Reverse a List with Collections.sort & Collections.reverse

* Collections.sort(listName); //void meaning you don’t have to assign this into a variable, it changes the list for you from least to greatest
* Collections.sort(listName); //void meaning you don’t have to assign this into a variable, it changes the list for you greatest to least

# 49. Nested for each loop

* List<List<Integer>> listNames = new ArrayList<List<Integer>>;
  + listNames.add(Arrays.atList( Arrays.asList(5, 3, 2, 6, 3));
  + listNames.add(Arrays.atList( Arrays.asList(5, 4, 2, 7, 8));
  + listNames.add(Arrays.atList( Arrays.asList(0, 3, 9, 2, 1));

| for(List<Integer> listName : listNames) {  for(int variable : listName) {sysout(variable);}  } |
| --- |

# 48. For each Loop in Java

* for(int variable : listName) {sysout(variable);} //prints the list as a variable

# 47. For Loops with Lists and how to modify each element

* listName.set(index#, value) //sets a new number for that index
* listName.get(index#); //returns the element in that index

# 46.Quickly Initialize a List with Elements & How to Print List

* List<Integer> listName = Arrays.asList(5, 3, 2, 6, 3);
* listName.toArray(); // converts a list to an array that can be system out

# 45. Working with List (List Methods)

* listName.add(5); //saved in index 0
* listName.add(10); //saved in index 1
* listName.add(1, 7); //7 is saved in index 1 and 10 is now shifted to index 2
* listName.indexOf(7); //returns the index 1, returns -1 if the element doesn’t exist
* listName.contains(7); //returns true if 7 is in the array
* listName.isEmpty(); //returns true is the array is empty
* listName.remove(0); //removes the number at 0 index and the elements are shift down 1
* listName.clear(); //removes, clears the elements in the list

# 44. List Interface and ArrayList Implementation

* Example

| List<Integer> grades = new ArrayList<Integer>(); //List is an interface. ArrayList is an implementation |
| --- |

# 43. ArrayList Introduction

* ArrayList <dataType> arrayName = new ArrayList <dataType>();
* arrayName.add(5); //adds 5 into the array
* arrayName.get(0); - returns with 5
* arrayName.set(0, 10); - changes index 0 of 5 to 10
* arrayName.size(); - returns with how many elements are in the array
* <> - a generic, generic programming allows you to create a class that can work with various types

# 42. Iterate through 2D Structure with a for Loop

* arrayName [0].length; - returns the number of elements are in row 0
* Examples

| int [ ] [ ] grades = {  {1, 5, 3},  {8, 4, 2, 4, 3, 5, 3},  {4, 6, 2}  };  for(int row = 0; row < grades.length; row ++){ //grade.length returns the number of the least row with elements  for(int col = 0; col < grades [row].length; col++){  sysout(grades [row] [col] + “ “);  }  sysout();  } //Outputs “1 5 3  8 4 2 4 3 5 3  4 6 2” |
| --- |

# 41. Working with 2D Arrays

* Examples

| int [ ] [ ] grades = new int [5] [5];  grades [0] [1]= 5; //Assigns that coordinate the number 5  int [ ] [ ] grades = {  {1, 5, 3},  {8, 4, 2, 4, 3, 5, 3},  {4, 6, 2}  };  sysout(grades [1] [1]); //Outputs the number 4  sysout(Arrays.deepToString(grades)); //Outputs the array like a list  [[1, 5, 3], [8, 4, 2, 4, 3, 5, 3], [4, 6, 2]]int [ ] [ ] grades = new int [5] [5];  grades [0] [1]= 5; //Assigns that coordinate the number 5  int [ ] [ ] grades = {  {1, 5, 3},  {8, 4, 2, 4, 3, 5, 3},  {4, 6, 2}  };  sysout(grades [1] [1]); //Outputs the number 4  sysout(Arrays.deepToString(grades)); //Outputs the array like a list  [[1, 5, 3], [8, 4, 2, 4, 3, 5, 3], [4, 6, 2]] |
| --- |

# 40. 2D Arrays

* 1D arrays are just a single line table
* 2D arrays are more than one table with multiple rows and columns
* dataType[ ] [ ] arrayName = new dataType [rows] [columns];
  + int [ ] [ ] arrayName = new int [3] [4];

arrayName [1] [2] = 5

| 0, 0 | 0, 1 | 0, 2 | 0, 3 |
| --- | --- | --- | --- |
| 1, 0 | 1, 1 | 1, 2 = 5 | 1, 3 |
| 2, 0 | 2, 1 | 2, 2 | 2, 3 |

# 39. Array Methods

* Arrays.fill(arrayName, value); - fills in the array with value
* List<dataType> variable = Arrays.asList(arrayName); - different way to sort an array
  + Need import java.util.List;
  + Need import java.util.ArrayList;
* Arrays.equals(arrayName1, arrayName2); -returns true if arrayName1 and arrayName2 equals each other

# 38. Arrays sort and Arrays parallelSort

* Arrays.sort(arrayName); - it doesn’t return anything it only modifies the array from lowest to highest
* Arrays.paralleleSort(arrayName); - it breaks the array into subarrays that are themselves sorted and they are merged together

# 37. Search an Array with a for loop

* Examples

| int[ ] arrayName = {1, 2, 3};  for(int i = 0; i < grade.length; i++){  If (grades[i] == 72){  sysout(“found”);  }  } //outputs nothing because 72 is not a stored value in the array  int[ ] arrayName = {1, 2, 3, 72, 5};  for(int i = 0; i < grade.length; i++){  If (grades[i] == 72){  sysout(“found at index ” + i);  }  } //outputs “found at index 3” because 72 is in the array |
| --- |

# 36. Array Values from input with a for loop

* Examples

| int[ ] arrayName = new int[10];  for(int i = 0; i <10; i++) {  grades[i] = 5;  }  sysout(Arrays.toString(arrayName)); //outputs “[5, 5, 5, 5, 5, 5, 5, 5, 5, 5]”  int[ ] arrayName = new int[10];  Scanner input = new Scanner(System.in);  for(int i = 0; i <10; i++) {  int x = input.nextInt();  grades[i] = x;  }  sysout(Arrays.toString(arrayName)); //outputs “[the x input 10 times]”  Scanner input = new Scanner(System.in);  Int size = input.nextInt();  int[ ] arrayName = new int[size];  for(int i = 0; i < size; i++) {  int x = input.nextInt();  grades[i] = x;  }  sysout(Arrays.toString(arrayName)); //outputs “[the x input size number of times]” |
| --- |

# 35. Arrays toString and Arrays deepToString

* Examples

Int [ ] arrayName = {9, 9, 7, 4, 3, 1, 4, 2, 5, 3};

Arrays.toString(arrayName); - returns the array list

Output “[9, 9, 7, 4, 3, 1, 4, 2, 5, 3]”

Arrays of arrays = Int [ ] [ ] arrayName = {{9, 9, 7}, {4, 3, 1, 4, 2, 5, 3}}

Arrays.deepToString(arrayName); - returns the array list of the array

Output “[[9, 9, 7], [4, 3, 1, 4, 2, 5, 3]]”

# 34. Working with Arrays

* Examples

| int[ ] grades = new int [10];  grades[0] = 10; //gives the first spot the number 10 to store  sysout(grade[0]); //shows the number of the 0 index which is 10  int[ ] grades ={9, 9, 7, 4, 3, 1, 4, 2, 5, 3};  sysout(grade[0]); //shows the number of the 0 index which is 9  grades[0] = 900; //updates the 0 index to 900  sysout(grade[0]); //shows the number of the 0 index which is now 900 |
| --- |

# 33. Intro to Arrays

* Array is a type of collection, a group of things

| int[ ] arrayName;  int[ ] arrayName = new int[10];  int[ ] arrayName = {12, 26, 3, 4, 5, 16, 7, 86, 95, 10}; //there are 10 elements that have a value  int arrayName [ ]; |
| --- |

* int [ ] is the data type int array, so it only stores int values.
* New int [10] makes the list have 10 elements. The index starts at 0 to 9
* Can’t go past the index of the array
* Import java.util.Arrays;
* Import java.util.ArrayList;
* Some functions:
  + arrayName.length; -returns the number of elements within the array.
  + arrayName[9]; -indicates the 9th index (the 10th element) of the array

# 31. Break and 32. Continue

* Continue statement is a way to jump to the next iteration of the loop
  + continue;
* Break statement breaks out of an inner loop.
  + break;

# 30. Variable scope with nested control flow

* You cannot declare a variable more than once if it is still in the scope

# 29. Nested while loops

* Example

| int i = 9;  while(i >= 0) {  int k = i;  while(k >= 0) {  System.out.print(k+ " ");  k--;  }  i--;  System.out.println();  } |
| --- |

# 28. Nested for loops (triangles and pyramids)

* Example

| for(int i = 9; i >= 0; i--){  for (int k = 9; k >= 0; k--) {  System.out.print((k+1)+ " ");  }  System.out.println();  } |
| --- |

* Example

| for(int i = 9; i >= 0; i--){  for (int k = i; k >= 0; k--) {  System.out.print((k+1)+ " ");  }  System.out.println();  } |
| --- |

# 27. Nested Blocks (nested if)

* Example

| if (i ==5){  if(variable == true){  //code  }  } |
| --- |

# 26. For Loops

* For loop is like a while loop
  + Initialization
  + Condition
  + Update
* Example

(will repeat 10 times)

| for(int i = 0; i < 10; i++){  //code  } |
| --- |

# 25. Do While Loop

* The do-while loop executes while loop at least once.
* Example

| do{  //code  } while(condition); |
| --- |

# 24. Intro to Loops (while loops)

* While loop - repeats until variable reaches the comparison. Works if comparison is true
  + Initialization
  + Condition
  + Update
* Examples

| Int i = 0;  while(i < 10){  //code  i++;  }  while (!guess.equals(password)){  sysout (“guess the password”);  guess = input.nextLine();  } |
| --- |

# 22. Ternary Conditional Operator

* ? ternary operator. Like and if-else statement
* Examples

| boolean welcome = name.equals(“clare”) ? true : false;  //if the variable name == clare then the statement will return true  //if the variable name != clare then the statement will return false  int points = name.equals(“clare”) ? 4 : 6;  //if the variable name == clare then the statement will return 4  //if the variable name != clare then the statement will return 6 |
| --- |

# 21. Switch Statement

* If statements are good if you have ranges. Switch statements are good if we can choose from a discreet number of values.
* The default acts as a catch-all and must be at the bottom.
* You need a break statement or it's going to run the next case until it reaches a break.
* You can have many cases.

| switch(variable){  case “variable 1 value”:  case “variable 2 value”:  //code // if you have “variable 1 value” or “variable 2 value” it will run “variable 2 value” code.  Break;  default:  //code  break;  } |
| --- |

# 20. Comparison and Logical Operators

* Comparison operators (returns true or false)
  + == equality
  + != unequal
  + < less than
  + > greater than
  + <= less than or equal to
  + >= greater than or equal to
* Logical Operators
  + && and (both have to be true)
  + || or (either can be true)
  + ! not (inverse the boolean)

# 19. If, Else If, Else

* Use if to specify a block of code to be executed, if a specified condition is true
* Use else to specify a block of code to be executed, if the same condition is false
* Use else if to specify a new condition to test, if the first condition is false

| if(condition){  //code  } else {  //code  }  if(condition){  //code  } else if(condition) {  //code  }  Else{  //code  } |
| --- |

# 18. String Comparison and Interning

* var1 == var2 comparison operator. This works for primitives and doesn’t work for objects. Values of objects are a reference to the object. Holds true if both variables have the same location of memory
* var1.equals(var2) - returns true if both variables hold the same thing. Use this one.

# 17. Creating Basic Classes, Methods, and Properties

* + Classes are often used to set the structure for something you might want to represent
  + Members - methods and/or properties
* Example

| public class program{  public static void main(String[] args) {  User user = new User();  user.firstName = “Dakota”; - setting a property  user.lastName = “Gullicksen”;  System.out.println(user.getFullName);  }  }  public class User{  public String firstName;  public String lastName;  public String getFullName(){  return firstName +” “ + lastName;  }  } |
| --- |

# 16. More String Methods (toLowerCase, strip, substring, repeat, equals)

* String methods (all start with a variable name)
  + .toUpperCase() - returns all letters in the variable uppercase
  + .toLowerCase() - returns all letters in the variable lowercase
  + .strip() - returns variable with no unneeded spaces, tabs, or new lines
    - .stripLeading() - gets rid of the beginning spaces, tabs, or new lines
    - .stripTrailing() - gets rid of the ending spaces, tabs, or new lines
  + .substring(starting index) - returns the variable starting at the index
    - .substring(start index, end index) - returns the variable starting at the start index and ending at the end index
  + .repeat(value) - repeats the string value number of times
  + .equals(variable2) - returns true if variable = variable2

# 15. String Methods (charAt, concat, contains, indexOf, lastIndexOf)

* String methods (all start with a variable name)
  + .charAt(index) - strings start at 0 - returns letter
  + .concat(argument) - returns what was in the variable with the added argument
  + .contains(arguement) - returns true if argument is in the variable
  + .indexOf(argument) - returns a numeric value of the argument location within the string
    - .indexOf(argument, index) - returns a numeric value of the argument location within the string starting at location index
  + .lastIndexOf(argument) - returns a numeric value of the last location of the argument within the variable

# 14. String Class (String.format, length)

* String class
  + String - a list of characters in “” quotes
  + .length() - returns number of characters in a string
  + .format(“%s”, variable) - returns string before %s and adds the variable
* Special characters
  + \n - new line
  + \’ - single quote mark
  + \” - double quote mark
  + \t - tab
  + \b or \\ - backslash
  + \r - carriage return
  + \f - formfeed

# 13. Numeric Methods (max, compare, valueOf, parseInt, etc)

* Integer.max(x, y) - returns the biggest number from the variables
* Integer.compare(x, y) - returns -1 if x is smaller returns 1 if y is smaller

# 12.Numeric Expressions and Operators

* int a;
* double x = (double) a / 2; - returns 2.5
* + addition
* - subtraction
* / division
* \* multiplication
* % modulus - gives us the remainder
* precedence - order evaluated
  + Example: ((5+3) / 2) \* 3 = 12
* Increment operator - variable++; after adds 1 to variable
* Decrement operator - variable--; after subtract 1 to variable
* Prefix Increment operator - ++variable; first adds 1 to variable
* Prefix Decrement operator - --variable; first subtract 1 to variable
* variable += number - variable = number + variable

# 11.Numeric Data Types and Properties (Infinity, NaN)

* datatype.SIZE - returns byte size
* datatype.MIN\_VALUE
* datatype.MAX\_VALUE
* Double.POSITIVE\_INFINITY
* Double.NEGATIVE\_INFINITY
* Double.NaN = Not a Number
* Double.isNaN(f(x)) - returns true if f(x) is NaN

# 10. Scanner input

* import java.util.Scanner;
* Scanner scan = new Scanner(System.in);
  + scan.next(); - allows input for a one string word
  + scan.nextLine(); - allows input for a line of string
  + scan.nextInt(); - allows input for a int
  + scan.nextDouble(); - allows input for a double
  + scan.nextByte(); - allows input for byte
  + scan.nextShort(); - allows input for short
  + scan.nextLong(); - allows input for a long
  + scan.nextFloat(); - allows input for a float
  + scan.nextBoolean(); - allows input for boolean
  + scan.close(); - ends the use of the scanner

# 9. Primitive Data Types

* objects - an instance of a class (class the data type)
* statically typed - variables are given data types upfront (before compiling)
* dynamically typed - variables do not have types
* date type - the type of the data. How the computer interprets the data
  + primitive types - boolean, byte, char, short, int, long, float, double
    - boolean - true or false
    - byte - 8 bit number or character the computer reads in ones and zeros
    - char - a single character in ‘’ quotes
    - short - 16 bit number the computer reads in ones and zeros
    - int - 32 bit number the computer reads in ones and zeros
    - long - 64 bit number the computer reads in ones and zeros (end with an L)
    - float - 32 bit decimal number the computer reads in ones and zeros (end with an f)
    - double - 64 bit decimal number the computer reads in ones and zeros

# 6. Variables

* variable declaration
  + 1. declare
  + 2. Initialize
* literals - the value
* variable - store some value
* expression - evaluates to a value
* operator - work on operands to produce a value
* operand - the thing(s) the operator works on

# 3. Understanding Java

* //comment
* class - contains everything - has members
* methods - do something
* arguments - what you pass to a method (part of the calling)
* parameter - Variables to store arguments (part of the definition)
* statements - telling the computer something
* properties - store something
* Access modifier (public) - who can use it?
* static - no instance of a class is needed
* abject - an instance in class

Start Date: 1/15/2022