Unit 1 - Getting Started, Primitive Types

Unit 2 - Using Objects

2.1 - 2.2

- Objects are variables of a class or reference type
- Class - a template for a new type of object
- Object instance - combines data and methods
- A constructor is a method that initializes the attributes of an object, they use keyword new
- public static void main (String[] args);
    - Sprite player = new player(30, 50);
- An overloaded constructor is when its a different constructor with the same name
- publicSprite() or publicSprite(double x, double y)
- If it has no parameters its called the default constructor
- Int x = 3 is a primitive type while Sprite player1 = new Sprite(100, 200) is a reference type
- You can initialize an object to null and if you try to use it you will get a null pointer exception

2.3 - 2.5

- Non-static or instance methods are part of a object an called with a dot operator and object name
- Static methods are part of a class and called with a dot operator and class name
- The driver class is the class with the main method
- Non static methods are usually implemented in object classes rather than driver classes (static)
- A void method doesn't return anything
- public static int method1
- public static void method1
- Value semantics - just cuz u changed it in the method, does not mean you changed it elsewhere
- Int to a double but not double to int

```
public class MyClass{
    public static void main(String[] args){
        System.out.println(SomeClass.method1());
        SomeClass a = new SomeClass();
        System.out.println(a.method2());
    }
```

```
}
public class SomeClass{
        public SomeClass()
        {…}
        public static int method1() // static method
        {…}
        public int method2() // non-static or instance method
        {…}
}
public class SomeClass{
        public SomeClass()
        {…}
        public static int method1() // static method
        {…}
        public int method2() // non-static or instance method
        {…}
```

2.6 - 2.7

- \\ prints a backlash
- \n goes to a new line
- \t tabs
- \” prints a quote
- Int length()
- Int indexOf(string part)
- String substring(int start)
- String substring(int start, int end)
- String toUpperCase()
- String toLowerCase()
- name.substring(2, 4);
- Index starts at 0, if it isn’t found returns -1
- x.equals(y) returns true if x = y
- x.compareTo(y) returns - if x<y, + if x>y, = if x=y

2.8 - 2.9

- Math.random formula: (high-low) * Math.random + low
- Wrapper classes

Unit 3 - Boolean Expressions & If Statements

3.1, 3.5 - 3.6

- ==; equal to
- !=; not equal to
- ＜; less than
- ＞; greater than
- ＜=; less than or equal to
- ＞=; greater than or equal to
- = sets the value; == evaluates if they are equal
- ||; or
- &&; and
- !; not
- DeMorgan's Theorem !(A && B) == !A || !B; !(A || B) == !A && !B, remember that demorgan's changes everything, including the cases; they should switch too
- PNAO order of operations, parenthesis, not, and, or
- check the problematic one first

3.2 - 3.4

```
System.out.println("What temperature is it?");
double temp = keyboard.nextDouble();
if(temp> 90){
        System.out.println("Wear shorts!");
}else if (temp >70){
        System.out.println("Wear short sleeves!");
}else if (temp>60){
        System.out.println("Wear long sleeves!");
}else{
        System.out.println("Wear a jacket!");
}
```

3.7

- == compares the whole thing, the reference; primitives like booleans, ints, and doubles
- .equals compares the contents; strings and classes that u make

Unit 4 - Iteration

4.1

- A while loop has one condition and a body
- While (condition) {

Statement;
Statement;
}

- It continues running while the condition inside is true
- **While (again.equals(“y”){**
  System.out.println("Enter Y to run again, or N to quit");
  again = input.next();//allow the user to quit the program
  }
- Error checking - while (user input is bad){
  Tells user why it;s bad, gives a chance to enter input again
- break; //terminates the loop

4.2

- 4 parts in a for loop: initialization, termination, increment, looped command(s)
- **for(initialization; termination; increment){**
- **command(s);**
- **}**

4.3

- String algorithms; using string function such as .length() and .substring(), we can use for and while loops to iterate through a string

4.4 - 4.5

- Nested loops will check if outside is true, go all the way through the inside, then check outside again, and go through inside again

```
for(int r=0; r<3; r++)
{
   for(int c=9; c>7; c--)
     System.out.print(r+":"+c+" ");
   System.out.println();
}
```

r	c
0	9
	8
	7
1	9
	8
	7
2	9
	8
	7
3	STOP

output

0:9 0:8
1:9 1:8
2:9 2:8

- Be careful what variable you use, if you use the wrong one you could create an infinite loop
- Computational complexity: amount of math operations needed to do algorithm
- Total number of operations = total number of times the loops run

Unit 5 - Writing Classes

5.1

- Object data fields should be declared as private to ensure that it can't be accessed elsewhere (encapsulation)
- Setters - change the state of a data field
- Getters - return the value of a data field
- .toString(), print out an object

5.2, 5.4 - 5.5 (Vocabulary)

Accessor/Getter	Method that returns the current value of a specified data field.
Arguments	Values that are passed into a method or constructor via the heading.
Class	The code that describes the attributes and behavior of an object.
Constructor	A block of code that initializes an object's data fields when the object is created
Default Constructor	A constructor that has no arguments sent to it.
Instance Variable/Data Field	A variable used to track the state of an object. Generally "global" in scope, in that they can be accessed from anywhere within the same class.
Instantiation	The creation of an instance of an object. Accomplished by utilizing the keyword new, followed by a call to a constructor.
Local Variable	A variable that exists only within the method, constructor, or control structure in which it is declared.
Mutator/Modifier/Setter	Method that allows us to modify the values of instance variables.
Null	A reference to an object that has not been instantiated.
Object	An instance of a class (created in our drivers)
Primitive Type	Stores a single value with no built-in methods. For example: int, double, or boolean

5.3, 5.6 - 5.7

- Pre condition - what the method needs to work effectively
- Post condition - describe what the method does (output)
- Static methods belong to a whole class, only one version; can be called without creating an instance of a class
- Instance methods belong to an object, each object has a version; called using the object name . method

5.8 - 5.9

- Variables only exist within the structure they are defined in
- In a non-static method you can use the keyword this to reference the current object
- Shadowing is when you use the same variable name

```
public class Point {
        private int x;
        private int y;

        ...

        public void setLocation(int x, int y) {
           this.x = x;
           this.y = y;
        }
    }
```

- *Inside setLocation,*
  - *To refer to the data field x, say this.x*
  - *To refer to the parameter x, say x*
- Java thinks you are talking about the variable that you are closest to

Unit 6 - Array

6.1

- Single entity for storing many of the almost same things
- String[] names = new String[//number of things you want//]
- When calling .length, no parenthesis; ex. names.length
- Default values: boolean, false; int, 0; double, 0.0; string, null

6.2

```
int [] nums = {5,9,7};          nums
int [] list = nums;
```

- If you do this the array is not copied, it becomes the same array; nums and list are now the same thing; changing one will change the other
- To copy an array you need to create a for loop and set every element equal to the other array's elements
- You will get a Null Pointer Exception if you don't set all the elements before you access them

6.4

```
Scanner input = new Scanner(System.in);
```

```
File reader
```

6.5

- Enhanced for loops are Compact notation to traverse from the first element to the last, can't be used to change anything, alter something or access the index

```
for(String x : names)
```

- "Iterate through all the String objects in names. Each time you go through the loop, x represents the next available String"
- [low, high) use (high - low) * Math.random() + low
- To include the high, add 1
- For an int cast it: (int) ((high - low) * Math.random()) + low

Warm Up

What is the output of the following code:

```
int[] list1 = {1,2,3,4,5};
int[] list2 = list1;
list2[0] = list1[0] + 10;
list2[3] += 1;
list2[1] = list1[0] - 5;
for(int n : list1){
    System.out.print(n + " ");
}
```

11 6 3 5 5

Check out slide 21 from the Day 2 Notes

- Creating aliases
- If you set the array equal to the other if you change one array the other one also changes

Unit 7 - Array List

Intro

```
boolean add(AnyType x)          //adds x to end of list, returns true if
                                //successfully added, false otherwise
```

```
void add(int index, AnyType x) //adds x at index, shifting elements right

AnyType remove(int index)          //removes and returns element at index
                                   //shifting remaining elements left.
int size()                         //returns number of elements.

AnyType get(int index)             // returns the element at index.

AnyType set(int index, AnyType x) // changes the element at index with x
                                  // returns old element.
ArrayList <Integer> nums = new ArrayList();
```

- Default arraylist size is 10
- It hold buffer space and if it runs out it will increase to double the current size
- If logical size is less than ⅓ the actual size the buffer space will cut in half
- BINARY SEARCH take log2(n) comparisons at the most for a list with length n

Selection Sort

- Find the smallest
- Swap it with 0
- Find the smallest from 1 to end
- Swap with 1
- Find smallest from 2 to end
- Swap with 2
- In the end it looks like a staircase going down, it starts becoming correct from the left and stays correct all the way down to the right
- You swap two elements in a selection sort

Insertion Sort

- Sort subarray with the first 2
- Sort subarray with the first 3
- Keep going
- Also looks like staircase going down but the numbers at the back will always be the same before its their turn to be switched
- Sorting subarrays
- Inserts the element at the correct position and everything gets shoved back

Bubble Sort - compare 2 swap, compare 2 swap

Quick Sort - fastest sort besides merge

Unit 8 - 2D Array

```
for(int r = 0; r < chart.length; r++)
{
    for(int c = 0; c < chart[0].length; c++)
    {
        System.out.print(chart[r][c] + " “);
    }
    System.out.println();
}
```

- Uses .length, no parenthesis

Unit 9 - Inheritance

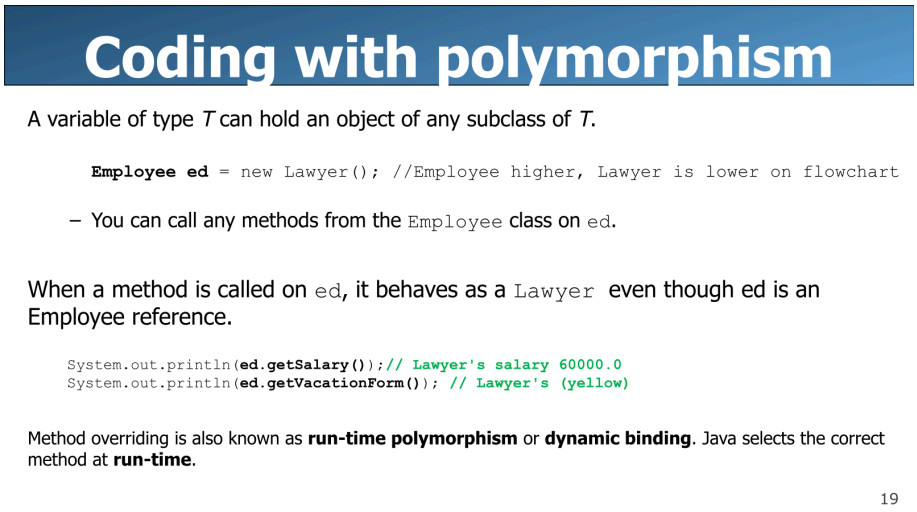
Inheritance

```
public class Precious extends Rock {
    - Public Precious(String c, double w, double v){
        - super(c, w);
        - value = v;
    }
}
return super.toString() + x + y;
```

- Uses super keyword to call things from the superclass
- Extends keyword to create the sub class

Polymorphism

- Lawyer can not use sue because it was declared as an employee



- By using super with no parameters you can call a method with no parameters
- Method overriding: run-time polymorphism
- Method overloading: compile-time polymorphism

Unit 10 - Recursion

Recursion

- Must have a base case and a recursive call

Merge Sort

- Break the array into 2 sub arrays
- Continue breaking down until each sub array has only 1
- Compare 2 and add smaller one to the array
- Continue comparing and adding until array is sorted

AP Exam Structure/Tips

MCQ

- 40 questions : 1 hr 30 min : 50% Exam Score

FRQ

- 4 questions : 1 hr 30 min : 50% Exam Score

1. Methods and Control Structures
2. Classes
3. Arrays/ArrayLists
4. 2D Arrays

- Make sure you know how to define arrays, arraylist, 2d arrays and classes

```
Int [][] mat = {{2, 3, 4, 5},
                {6, 7, 8, 9, 10},
                {11, 12, 13, 14}};
```

mat [2] = mat [0] ← this makes the third row a reference of the first row

Mat [0][0] = 1 ←so the third row also changes

Systemprint mat[1][1] * mat[2][0] ←2, 0 was changed so its 7 not 14

Merge sort has the best run time, merge sort is fastest

Switch, case →you keep going until you get to break

Arrays.sort → it just happens, you dont need to set the array equal to the sorted array because since its part of the array class it automatically happens

Run through the entire code when its asking what the output will be, dont skip anything or take shortcuts especially when its switching things or inserting things in sorts

ARRAY VS ARRAY LIST AND 2d ARRAYS KNOW everything
String [] nums = new String [3]
nums[0]
ArrayList <String> nums = new ArrayList()
nums.get(0)
String [][] nums = new String[3][3]

YOU HAVE TO RUN A FOR LOOP TO FILL AN ARRAY OR 2D ARRAY, you cant use add like in arraylist or just add it on to the end like a string

Sorts
Bubble
Selection

- Search for smallest put at 0, search for smallest between 1 and end, put at 1

Insertion

- Sort 2 subarray, sort 3 subarray… all the way down to all sorted

Quick
Merge, recursive

- Break down into length 1 subarrays, compare subarrays and merge them together

Math.random (int) ((high - low + 1) * Math.random()) + low
Modulus; 3%5 is 3 because goes in 0 times, 3 left over

If this string comes first everything afterward is a sting so adding variable after wouldn't actually add them but just append the number that the variable is