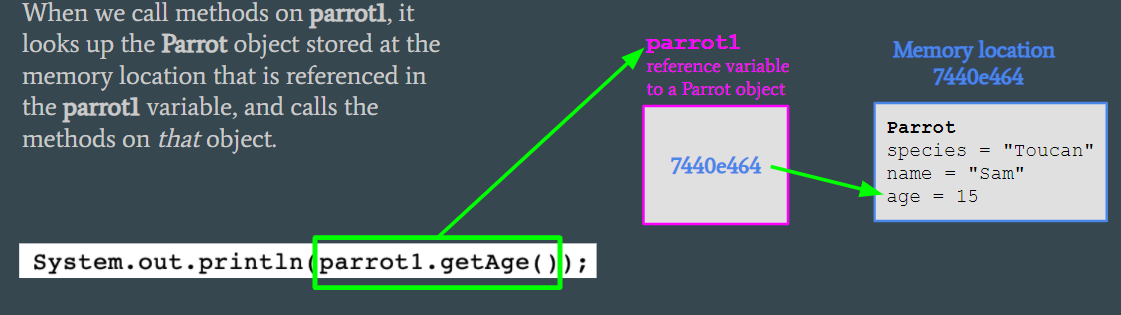
Intro Stuff

* Comments
  + //this is a single line comment
  + /\*
  + This is a multi-line
  + Comment
  + \*/
  + /\*\*
  + \* this is a multi-line
  + \* documentation
  + \*/
  + Comments do not run
* A program must contain at least one class, each containing at least one method, each containing at least one statement. \*One\* method must be named main, with signature public static void main(String[] args)
  + Args receives arguments through the command line when starting the program
  + Main method is that starting point
* JDK is the SDK for Java. It compiles human code into bytecode (which is sent to the JVM to compile to machine code for the target machine)
* JRE is the runtime environment for Java
* Syntax rules: rules enforced by Java the language.
  + Blocks of code surrounded by {}
  + Dot notation for OOP
  + Strings go in “”
  + Array elements go in []
  + Parameters and arguments listed in ()
  + Java keywords all lowercase
  + Statements end in ;
  + These type of errors are usually caught by the compiler (Compiler errors)
* Style rules: rules liked by humans for neatness
  + Indenting. Real-world usually uses 4 spaces, but APCSA uses 2
  + Capitalizing the first letter of the name of classes
  + Surround operators with one space each direction
  + Names of constants are all in caps, and \_ is used as space
  + camelCasing
  + Real-world uses braces like this:
    - Something {
    - }
  + APCSA does this
    - Something
    - {
    - }
* Runtime errors: errors that occur during runtime. Not syntax errors, because otherwise, the compiler will have caught it
* Documentation: [Overview (Java SE 15 & JDK 15) (oracle.com)](https://docs.oracle.com/en/java/javase/15/docs/api/index.html)
* The only things that exists on the AP: [AP Computer Science A Java Quick Reference (collegeboard.org)](https://apcentral.collegeboard.org/pdf/ap-computer-science-a-java-quick-reference.pdf?course=ap-computer-science-a)

Data Types

* Primitive
  + Numerical
    - Integers
      * Short - doesn’t exist in APCSA - 16 bits
      * Int - 32 bits
        + Max number is +/- (2^32 - 1). 1 is subtracted because one bit is a signed bit
        + Use under\_score instead of commas if you want commas
        + If you go past the max in an arithmetic operation, the operation still gets performed, but overflowed bits get dropped (wrap around)
        + If you cast a double that’s numerically larger than the int max, the output is the max int value
        + Integer.MAX\_VALUE is the largest value
        + Integer.MIN\_VALUE is the lowest value
        + Wrapper class: Integer
      * Long - doesn’t exist in APCSA - 64 bits
    - Floating-point decimal
      * Float - doesn’t exist in APCSA - 32 bits
      * Double - 64 bits
        + Wrapper class: Double
  + Textual
    - Byte - doesn’t exist in APCSA - 8 bits
    - Char - doesn’t exist in APCSA - 16 bits
  + Boolean - 1 bit
    - Wrapper class: Boolean
  + A primitive type’s variable’s memory space would directly store the value itself
* Reference Type (Objects)
  + Classes
    - One class is stored in one file
    - Filename: classname.java
    - Class contains methods (functions belonging to the class)
    - One class has a method named main, which is the main function. (The thing JRE automatically runs when app is opened)
    - JDK compiles \*.java (human code) into bytecode (computer code) \*.class file
    - Keywords:
      * public: accessible globally in Java app
      * private: accessible only in the class it’s declared in
      * class: indicates a class is being defined
      * void: method does not return a value
      * static: accessed through the class, not instances of the class
        + Cannot access instance variables and methods
    - Defining a class: sample
      * HelloWorld.java
      * public class HelloWorld {
      * //methods in here
      * }
    - Defining a method: sample
      * public static void main(String[] args) {
      * //code to run goes here
      * }
  + The new keyword creates an object in some memory location. A reference type variable will only store a reference type, or the memory address for referencing the location of the actual object in memory.



* + - garbage collection automatically removes objects from memory if nothing can refer to it anymore
  + String
    - Characters between “”
    - Concat strings “first string” + “second string” = “first stringsecond string”
    - Can concat with numbers to result in a String
    - Strings could also be created new String(“something”);, but would lose the ability to use == to compare string contents
    - Escape character (\)
      * \\ to type a \
      * \” to type a “
      * \n for a new line
    - Methods
      * int length(): returns the length of this String as an int
      * int indexOf(String string): returns the first index where string starts from in this String, as an int
        + If string isn’t found in this String, -1 is returned
      * String substring(int start, int end): returns a String that’s based on this String, starting from index start, up to index (end - 1)
      * String substring(int start): returns a String that’s based on this String, starting from index start, up to the end
      * boolean equalsTo(String other): returns if this String and other String contains the same characters in the same order
      * int compareTo(String other): let's say this String and other String gets put in alphabetical order (technically, it’s based on unicode number). Alphabetically, how far is this String from the other String? That’s what gets returned.
        + Ex: “base”.compareTo(“cat”) == -1
        + “” (emptiness) comes before everything
        + Uppercase characters come before lowercase characters
        + Now, what if the first character of both Strings are the same? Then it would be based on the next character. If that’s also the same, then base on the next character, and so on. If everything's the same, return 0
      * .toUpperCase returns the string all uppercased
      * .toLowerCase returns the string all lowercased
* System.out.println(literal) prints literal into the console, then add a new line
* System.out.print(literal) prints literal into the console without adding a new line
* If nothing is passed into println, just a new line is created
* Variables
  + To define a variable: start with the type, then the name, then = value
    - ex: boolean someBoolean = true;
  + When a variable is defined, the amount of memory needed for that data type is reserved
  + Block scoped
  + Default values
    - Int: 0
    - Double: 0.0
    - Boolean: false
    - Reference types: null
  + Variables are statically typed, so once a variable is given a type, it usually can’t store data of another type
    - Casting
      * Automatic
        + From less memory used to more memory used
        + byte -> short -> char -> int -> long -> float -> double
        + If a variable for numbers is declared in a type with more memory, it can store number data types that would’ve taken less memory
        + ex: you could store an int in a double variable
        + When evaluating arithmetic expressions, the result would be of the data type in the expression using the most memory
        + ex: adding a int (32 bit) to a double (64 bit) will result in a double (64 bit) answer
      * Manual
        + From more memory used to less memory used
        + Syntax: (destinationType) originalVariable
        + Ex: (int) someDouble

Casts the double someDouble into an int

The decimal part is dropped

* + Variable names can’t be Java keywords
  + Put final before the variable type to define the variable as a constant (can no longer write to the variable)
  + To update a variable’s value: variableName = newValue;
* User input
  + First, import java.util.Scanner
    - First lines in the file, outside of the class
  + Then, create an instance of Scanner
    - Scanner scan = new Scanner(System.in);
  + Then, to open command line to accept an input: <Scanner>.nextLine();
  + The nextLine() method returns a String from the user’s input
  + New line is created after user enters an input
  + (or nextInt, nextDouble, etc)
  + Add a nextLine after a nextInt or else unexpected behaviors may happen
* Operators
  + = set the variable on the left equal to the value on the right
  + + add
    - Can also concat strings
  + - subtact
  + \* multiply
  + / divide
  + % modulo
    - Remainder after dividing left by right
  + Putting = after +, -, \*, / or % means to perform the operation first, then store result to variable on left
  + PEMDAS applies
  + ++ adds 1
  + -- subtracts 1
  + == equivalent

Classes

* Class design
  + Classes by convention have names starting with a capital letter
  + Classes are in their own \*.java file
  + Class files don’t run themselves; they’re used by a “client class” (the one with the main method)
  + Classes to be used must be in the same directory as the client class (unless it’s one that’s included in the runtime)
  + Properties
    - Declare them first like in the sample below
    - private means can only be accessed within the class file
      * File as in an object passed into a method of another object of the same class, the dot operator can be used
    - public means can be accessed outside the file too, using dot notation on an instance of that class
      * instanceOfCat.furColor;
    - Accessing properties within the class file doesn’t need this
    - Instance variables (those belonging to a specific instance of a class) are kept private by encapsulation conventions
    - Adding static after public or private makes it a static variable
      * Public static variables can be reference through dot notation on the class name
      * Static variables can be accessed by both static and instance methods
      * Only one copy exists in memory
    - Adding final before the return type makes the variable a constant
      * Value cannot be modified after it’s set
  + Constructor
    - Defined within public ClassName(parameters)
    - Name must match the class name
    - Creates an instance of the class
    - There can be multiple constructors; when constructing an instance of the class, based on the alignment of the types of arguments passed in, the correct constructor will automatically be chosen
      * Called overloading
  + Methods
    - Defined in public (or private) returnType methodName(parameters)
    - public methods can be accessible anywhere by using dot notation on an instance of the class
      * instanceOfCat.feed();
    - private methods can only be accessed within the class file
    - Accessing methods within the class doesn’t need this
    - If the method doesn’t return anything, then use void for the return type
    - Can also be overloaded like constructors
    - Accessors: returns the value of instance variables
      * For booleans, naming convention is isVariableName
      * For everything else, the naming convention is getVariableName
    - Mutators: modifies the value of instance variables
      * Naming convention is setVariableName, and the parameter is newVariableValue
    - Adding static after public or private makes it a static method
      * Public static methods can be reference through dot notation on the class name
      * Static methods cannot access instance variables
      * Only one copy exist in memory
  + Sample:
    - public class ClassName {
    - //define properties
    - private String someProperty;
    - //constructor
    - public ClassName(String somePropertyValue) {
    - someProperty = somePropertyValue;
    - }
    - //define methods
    - public int someMethod(int parameters) {
    - return 123;
    - }
    - public void someMethod(double parameters) {
    - something;
    - }
    - }
* Creating instances of a class
  + ClassName instanceName = new ClassName(arguments);
  + Classes that come with the runtime need to be imported with import
    - java.lang.\* package is pre-imported by default
  + Use dot notation to access public properties and methods and use parenthesis to pass in arguments
* Javadoc
  + /\*
  + \* used to document a method
  + \* like a description for it
  + \* @param someParameter as well as explain what the parameters do
  + \* @return as well as what it returns
  + \* @author
  + \*/
  + Everything should be documented
* When printing an object literal, the object’s memory address is printed instead. If that object’s class has has a public String toString() method, then that method would execute instead (overridden)
* Math library
  + The Math class
  + static
  + Math.PI: pi
  + int abs(int x): returns the absolute value of x
    - Also has a double version
  + double pow(double base, double exponent): returns baseexponent
  + double sqrt(double x): returns the square root of x
  + double random(): returns a double between 0 and 1 (not including 1)
* this keyword
  + Inside an instance method, it refers to this instance of the class
  + Elsewhere, it represents the constructor for this class
* Wrapper classes
  + Classes that wraps primitives into reference types, for situations where a reference type is required
  + Integer, Double, Boolean, etc
  + Auto boxing: in a situation where a data of a reference type is required and the data supplied is of a primitive type, Java will automatically put that primitive data into its wrapper
  + Manual boxing: manually putting a primitive into its wrapper
    - example: Integer num = new Integer(21);
  + Auto unboxing: in a situation where a data of a primitive type is required but the data supplied is of a reference type, Java will automatically unwrap the primitive data out of its wrapper
  + Manual unboxing: manually taking the primitive data out of its wrapper
    - Example: num.intValue();
* Inheritance
  + A subclass inherits properties and methods from a superclass
  + To create a subclass, in the class heading, add the keyword “extends”, followed by the the superclass’ name
  + In the subclass’ constructor, start with super() (pass in the arguments for the superclass’ constructor) to initialize the properties it inherited (calls superclass’ constructor)
  + Note that the subclass can’t reference private properties and methods of the superclass
  + “Is-a” relationship: if ClassB is a subclass of ClassA, ClassB is a ClassA.
  + Unified Modeling Language diagram: has blocks each representing a class, with each block showing the properties and methods of that class (excluding the ones it inherited). Arrows drawn from a class to the class that it inherits from.
  + Overriding
    - To redefine a method that was inherited from a superclass, simply by defining a method with the same header as the method you’re trying to override
    - The @Override annotation tells the compiler to verify that the new method is actually overriding a method (ie you didn’t make a typo in the header)
    - Use super.methodName() to refer to the original definition
  + Polymorphism
    - If ClassB inherits from ClassA, then a variable declared as a ClassA type can store a ClassB type
    - Can be useful in cases like parameters and arrays
    - However, methods in ClassB that aren’t in ClassA can’t be resolved by the compiler
    - In order to resolve ClassB’s methods, the variable would have to be casted to ClassB

Selection and Logic

* Boolean: value that’s either true or false
* Operators
  + == equals
    - For strings, can only compare content if the string was made as a literal (ex: “i am a string”). Or else, need to use the equals method
    - For objects, it compares if both variables are pointing to the same memory location
    - Be careful with doubles, as floating point imprecision would lead to unexpected results
  + != not equal
  + < less than
  + <= less than or equal to
  + > greater than
  + >= greater than or equal to
  + && AND evaluates to true only if left and right expressions are true
  + || OR evaluates to true as long as at least one expression is true
  + ! NOT inverts the truth value
* Compound operators
  + Order of operations for compound operators is () → ! → && → ||
  + Short circuit evaluation
    - When evaluating an AND, if the left expression is false, that AND expression automatically evaluates to false, without evaluating the right expression
    - When evaluating an OR, if the left expression is true, that OR expression automatically evaluates to true, without evaluating the right expression
  + DeMorgan’s Law
    - If the value of an expression involving an operator is to be inverted, that is equivalent to distributing the NOT operator to both sides of that expression and inverting the operator
    - Ex: !(a || b) == !a && !b
    - && becomes ||
    - || becomes &&
    - == becomes !=
    - != becomes ==
    - > becomes <=
    - < becomes >=
    - >= becomes <
    - <= becomes >
* Structures
  + if (boolean expression) {
  + //code to run if true
  + } else if (boolean expression) {
  + //code to run if true
  + } else {
  + //code to run if nothing above ran
  + }
  + Brackets can be removed if the code to run is only a single line
* Ternary
  + (condition) ? returnValueIfTrue : returnValueIfFalse
* Switch
  + switch (switchValue) {
  + case caseValue:
  + //code if caseValue == switchValue
  + break;
  + case anotherCaseValue:
  + //code if anotherCaseValue == switchValue
  + break;
  + }

Iteration

* while (condition) {
* //code to run while condition is true
* }
* for (initialization; test condition; increment) {
* //code to run while condition is true
* }
* Transversal: to loop through every element/character
* For loops are typically used when you know generally what number of counts the loop should run, while while loops are typically for if you have a stopping condition
* break: ends the current loop

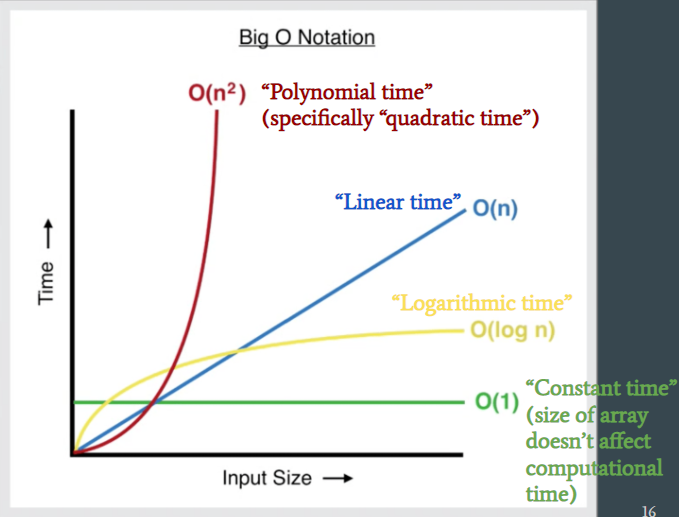
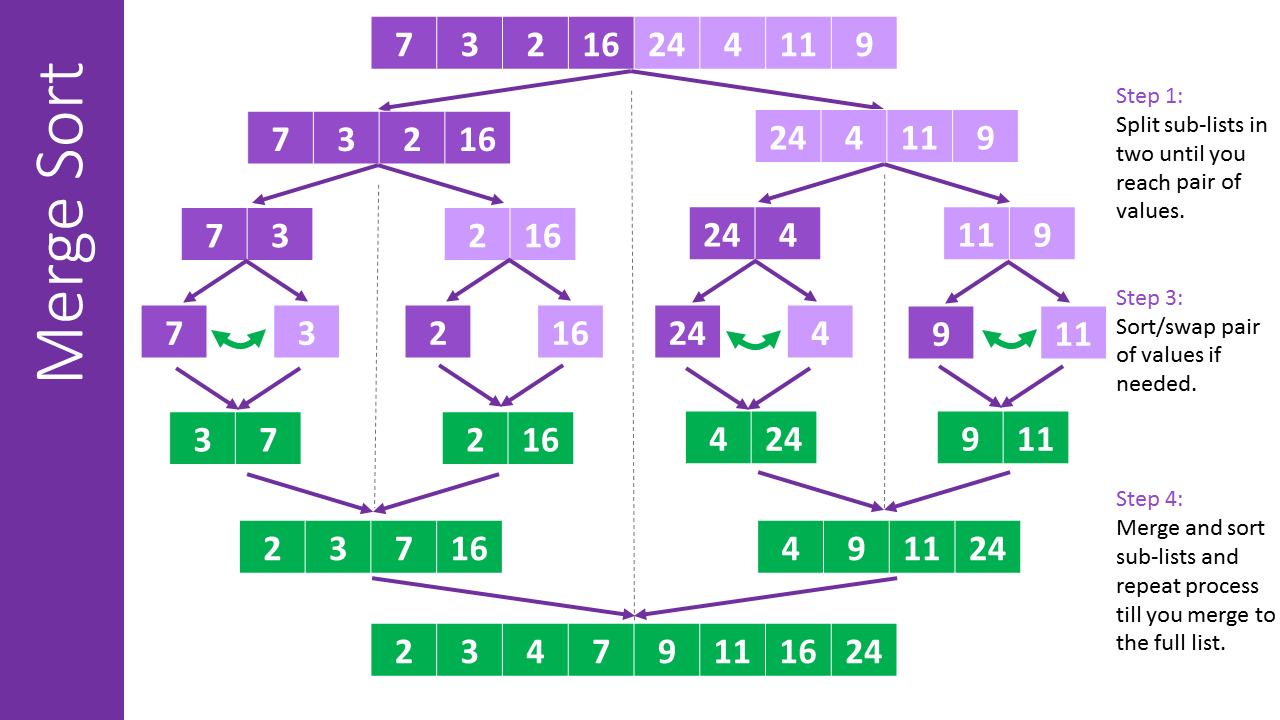
Array

* Arrays can only store the same data type in each of its indexes
* Array length cannot be modified once set
* To create an array
  + The data type would be the data type the array would store, with [] added to the end
  + To predefine values in the array, put them in { }, with elements delimited by a comma
  + To create an empty array with a certain number of indexes, use new followed by the data type used for that array, with the number of indexes inside the []
    - Each element would then be the default value for that data type
  + Ex: int[] predefinedArrayOfInts = {1, 2, 3, 4}
  + Ex: String[] emptyArrayOf5Strings = new String[5]
* Arrays are zero-indexed
* To reference an element at a certain index, use arrayName[index]
* The length property can be accessed using dot notation
* A useful library of Array utilities is java.utils.Arrays
  + Contains a toString method for Arrays
* Enhanced for loop
  + for (type variableName : arrayName)
  + This loops arrayName from index 0 to last index in sequential order
  + variableName is a copy of the current element in iteration
  + Modifying the value of variableName does not alter arrayName if primitive
  + Also works for ArrayLists
    - Cannot insert or remove elements from an ArrayList that is currently being iterated by an enhanced for loop
* 2D Array
  + An array of arrays
  + To create a 2D array
    - The data type would be just like what you would use for a 1D array, but add a second []
    - To create a 2D array with default values, it’s like 1D arrays, but the number that goes in the first [] is the number of arrays/rows to make, and the number that goes in the second [] is the number of elements/columns in each array/row
    - Ex: int[][] someArrayOfIntegers = new int[4][6];
      * A 2D array of Integers with 4 rows and 6 columns
    - To create a 2D array with predefined values, it’s like 1D arrays, but the contents of each row goes in it’s own {}, that’s inside the “parent” {}
    - Ex: boolean[][] someArrayOfBooleans = {{true, true}, {true, false}, {false, false}};
  + To reference a certain index of the 2D array, use arrayName[row number][column number]
  + All other rules about Arrays apply

ArrayList

* Dynamic in size
* Needs to be imported via java.util.ArrayList
* Can only hold reference types
* Can hold different types in different indices of the same ArrayList
  + Doing so would cause the elements become of type Object, making some methods unresolvable without downcasting
* Creating an ArrayList
  + All elements of the same type
    - ArrayList<E> name = new ArrayList<E>();
    - E is the type
  + Allow elements of different types
    - ArrayList name = new ArrayList();
  + To prefill elements
    - Import java.util.Arrays
    - Pass Arrays.asList(list, stuff, delimited, with, comma) as argument into the ArrayList constructor
    - If you already have an Array you want to convert into an ArrayList, pass that array into Arrays.asList(), which then gets passed to the constructor
      * Must be an Array of Objects
* Getting the size
  + name.size();
* Adding to an ArrayList
  + name.add(thing);
  + Returns true
* Replacing an element at a certain index with another element
  + name.set(index, newThing);
  + Returns the element previously at index
* Get element from a certain index
  + name.get(index);
* Contents can be printed using toString
* Adding an element to an ArrayList at a certain index, without overwriting anything
  + name.add(index, newThing);
  + newThing would be placed at index index, and everything that was already at index index and after it would be pushed down one index
  + void
* Removing an element from an index
  + name.remove(index);
  + Returns the element that got removed
  + All the elements after index index slides down one index to fill in the gap
* When traversing an ArrayList and inserting an element, increase the index counter by 1 afterwards
* When traversing an ArrayList and removing an element, decreases the index counter by 1 afterwards

Algorithms

* Complexity
  + Amount of resource needed to run an algorithm
  + Time complexity is about runtime (which means number of steps)
  + Space complexity is about memory
  + Big-O
    - Function modeling the amount of resources O required in worse-case scenario given the number of inputs n
    - 
    - A loop traversing something may be described using linear time
    - A loop traversing something while nested inside another loop traversing something may be described using quadratic time
    - Let's say the big-O notation for one algorithm is 2O(n) and O(n) for another. Both are still linear time, but one is “better” than the other
  + Big-Omega
    - Function modeling the amount of resources omega required in best-case scenario given n inputs
  + Big-Theta
    - Function modeling the amount of resources theta required in actual-case scenario given n inputs
* Searching
  + Linear search
    - Check consecutive elements one by one until the query is located
    - Best case: 1
    - Worst case: N
  + Binary search
    - Requires a sorted array
    - Start at the middle, and see if the middle element is higher or lower than or equal to the query
      * For even number of elements, traditionally, the left middle is used as middle
    - Then, eliminate the half that the query cannot be located in
    - Repeat with the remaining part, until the query has been located
    - Best case: 1
    - Worst case: log2(N)+1
      * Trick: 2number of steps = max number of elements
      * So if you have 15 elements, 23=8, so can’t be 3 steps, but 24=16 and 15<16, so it’s 4 steps
    - Average case: faster than linear search. Tends closer to worst case time
* Sorting
  + Selection sort
    - An outer loop traverses through the array
    - At each element, an inner loop checks all of the proceeding elements to identify the smallest value of those
    - Swap the current element (from the outer loop) with the the smallest value that the inner loop found
    - Worst case: polynomial time
    - Average case = worst case = best case
  + Insertion sort
    - An outer loop traverses through the array
    - At each element, an inner loop checks the preceding elements to identify how far back the current element (from the outer loop) should be sent back
    - Insert the current element back there
    - Worst and average cases: polynomial time
    - Best case: linear time
  + Merge sort
    - Utilizes recursion
    - 
    - It recursively breaks the array into halves until only pairs left, then swap element positions to sort the pairs, then merge sorted subarrays back together into a sorted array
    - Always has logarithmic runtime
    - If there are n elements, then the algorithm does n-1 merges
  + Array.sort
    - For objects, TimSort is used (a hybrid of Merge and Insertion sort)
    - For primitives, Dual-Pivot Sort is used
  + Bubble sort
    - Traverse through the array and compare adjacent elements. If the two elements are out of order, swap them.
    - Big O: n2

Other

* Graphical User Interface
  + Abstract Window Toolkit (AWT): java.awt.\*
    - To draw on a panel’s graphical context, call .getGraphics() on it and get the returned Graphics object
    - The Graphics Class
      * drawLine(x1, y1, x2, y2) to draw a line
      * drawOval(x, y, width, height) to draw oval
      * drawRect(x, y, width, height) to draw a rectangle
      * drawString(string, x, y) to draw text
      * fillOval(x, y, width, height) to draw a shaded oval
      * fillRect(x, y, width, height) to draw a shaded rectangle
      * setColor(color) to change the color of the “paintbrush”
      * setFont(font) to change the font of the “paintbrush”
    - The Color Class
      * Represents a color
      * Has static constants such as Color.ORANGE
      * Can create a custom color by doing new Color(r, g, b)
    - The Font Class
      * new Font(name, style, size)
      * Style constants
        + Font.BOLD
        + Font.ITALIC
        + Font.PLAIN
        + BOLD and ITALIC can be added
    - BufferedImage
      * You can create an image that you can get the Graphical context of to draw on
      * new BufferedImage(int width, int height, type)
      * Types:
        + BufferedImage.TYPE\_INT\_ARGB: transparent background
        + BufferedImage.TYPE\_INT\_RGB: solid black background
      * Import java.awt.image.\*
  + javax.swing.\*
    - JOptionPane
      * Shows a popup dialogue and give the user some options to react
      * JOptionPane.showMessageDialog(parentWindow, message): prints the message and provides an “OK” acknowledgement button for the user to click on
        + parentWindow can be null
      * JOptionPane.showConfirmationDialog(parent, message): like showMessageDialog, but has “Yes”, “No”, “Cancel” buttons, and the method returns ints JOptionPane.YES\_OPTION, JOptionPane.NO\_OPTION or JOptionPane.CANCEL\_OPTION depending on the button the user clicked on
      * JOptionPane.showInputDialog(parent, message): like showMessageDialog but provides a text box for the user to type a response into, and the method returns the String of what the user typed in
    - JFrame
      * A graphical window that can house components
      * <JFrame>.setVisible(true) tells the runtime to render the window on the screen
      * <JFrame>.setTitle(title) to title the window
      * <JFrame>.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE) to terminate the Java program upon closing the JFrame’s window
      * <JFrame>.setIconImage(Image) to set the window’s icon
      * <JFrame>.setLayout(LayoutManager) to set the window’s layout
      * <JFrame>.resizable(boolean) to set whether the window is resizable
      * <JFrame>.add(Component) to put the component onto the window
      * <JFrame>.pack() to make the window size as small as possible without cutting anything
    - JScrollPane
      * Container subframe for scrollable content
      * new JScrollPane(area)
    - JComponent
      * Superclass for GUI components
      * setBackground(Color): the background color
      * setEnabled(boolean): set whether the component can be interacted with
      * setFocusable(boolean): set whether the component can accept keyboard input
      * setFont(Font): set the font for writing
      * setForeground(Color): set the foreground color
      * setLocation(Point): set the coordinate of the top left corner of this component
      * setSize(Dimension): set the component’s size
      * setPreferredSize(Dimension): set the component’s preferred size
      * setVisible(boolean): set whether the component is visible
    - Dimensions
      * new Dimension(int width, int height)
    - JButton
      * new JButton(String text)
      * <JButton>.setText(String updatedText)
    - JTextField
      * new JTextField(int optionalLength)
      * The optional length is for how many characters can fit in the field
    - JLabel
      * A label
      * new JLabel(String message)
    - JTextArea
      * Textbox with multiple rows
      * new JTextArea(int rows, int cols)
    - Layouts
      * BorderLayout (default): by default components are centered in the window, takes up the full size of the window, and new components added are on top of existing components
        + Allows the add method to have a second parameter for where to place the Component

BorderLayout.NORTH, and other compass directions, including CENTER

* + - * + Sizes components to take up all spaces
      * FlowLayout: flow out components side by side
        + new FlowLayout();
        + Respects components’ preferred sizes
      * GridLayout: grids
        + new GridLayout(int rows, int cols)
        + Ignore components’ preferred sizes
      * Composite layout: using multiple layouts at one
        + JPanel: a container for components. Like a sub window you can add to a window

You can also draw on the panel

Create a class that extends JPanel

Override the paintComponent method

Be sure to call super.paintComponent in the first line of the new paintComponent method

Code for drawing goes after that line

* + - Events
      * import java.awt.event.\*
      * The class needing to handle events needs to implement the the listener in question
      * ActionListener: listener for when the user interacts with a component
        + public class YourGuiClass implements ActionListener
        + The class also needs to implement the the public void actionPerformed(ActionEvent event) method, which is the handler to run when the event happens
        + To attach your listener, <JComponent>.addActionListener(this)
        + The event argument contains data on the event
        + Timers

new Timer(int ms, ActionListener listener)

Calls the listener every ms milliseconds

<Timer>.start() to start

<Timer>.stop() to end

* + - * MouseInputListener: listener for mouse events
        + Handlers:
        + mouseClicked
        + mouseDragged
        + mouseEntered
        + mouseExited
        + mouseMoved
        + mousePressed
        + mouseReleased
        + Attaching listener

addMouseListener for presses

addMouseMotionListener for motion

* + - * + The event argument contains data on the event

getButton() for which mouse button was pressed

getClickCount() for how many times the mouse button is pressed

getPoint() to get the coordinates of the mouse

getX() and getY() for specific parts of the coordinate

* Interface
  + Describes what methods a class needs to implement in order to be considered to have implemented the interface
  + Like a superclass, but where each subclass’ method implements are expected to be different
  + A variable could be typed to accept objects of any class as long as a certain interface is implemented
  + To define an interface: interface InterfaceName
  + To implement an interface: public class ClassName extends SuperClassName implements InterfaceName
  + Include method headers but without bodies
* Abstract class
  + Like an interface, but can also include required instance variables
  + Add the abstract modifier to the class header
  + Write the class as normal, but any methods that subclasses should provide the implementation for should be left empty (no body) and include the abstract modifier
  + Subclasses extends abstract class like any other class, but just need to implement any methods labeled as abstract
* Call stack
  + Keeps track of which method is currently running and the “stack” of methods it was called from
  + When a method is called, it gets added to the stack
  + Once a method completes execution, it gets removed from the stack. Then, we resume running the method that called the method that just completed
  + There is a limit to how large the stack can be
* Recursion
  + A method that calls itself
* Inner classes
  + A class inside another class
  + The inner class is a member of the outer class
  + The inner class can access private members of the outer class, unless the inner class is declared static
  + An instance of the inner class can be constructed just like any other class inside the outer class
  + An instance of the inner class can also be constructed in classes other than the outer class
    - Outer outer = new Outer();
    - Outer.Inner inner = outer.new Inner();
    - This can be prevented by making the inner class private
  + Inner classes can also be placed inside of a method. It would only be constructible within that method
  + Use Cases
    - Logically groups together classes where a class would only be used in one class
    - Increase encapsulation
    - Improve code readability
  + Scopes
    - name would refer to in the current scope
    - this.name would refer to in the current class
    - OuterClass.this.name would refer to in the outer class