The keyword class announces that a new class is about to be named
Everything in java must be inside a class
Filename must match the public main class

Hello is the name of the class
this means the filename of the program must be Hello.java
The braces mark the beginning and end of the contents of class Hello
For each open brace there must be a corresponding closing brace

```
class Main
{
 public static void main(String[] args)
 {
   System.out.println("Hello world!");
 }
}
```

Output statements

- Can print one thing
    - Text enclosed in quotes
        - System.out.println(“anything you want to say”) ;
    - Any number
        - System.out.println(3.14159) ;
    - Can print multiple strings with +

      ```
      System.out.println("pi="+3.14159);
      Makes pi=3.14159
      ```
- println makes a new line
    - Use print to make without new line
    - `System.out.print("Hello world!");`

Syntax determines the structure

- Incorrectly written code (not put in properly)

Semantics defines the meaning

- Logic errors

- Class naming conventions
    - Name is an identifier consisting of alphanumeric characters
    - CamelCase
        - First letter is uppercase for each word, no numbers as start of word
        - For classes

Java generally ignores whitespace (tabs, newlines, and spaces) as long as it is not in between quotes " ".

\t
Inserts a tab in the text at this point.
\n
Inserts a newline in the text at this point.
\"
Inserts a quote in the text at this point.
\'
Inserts an apostrophe in the text at this point.
\\
Results in a single \ being displayed

8 primitive built in types of data

- boolean
    - True or false
- Byte
    - 8 bits
    - -2^7 to 2^7-1
- Short
    - 16 bits
    - -2^{15} to $2^{15}-1$
- int
    - Integer
    - 32 bits
    - -2^{31} to $2^{31}-1$
- long
    - 64 bits
    - -2^{31} to $2^{31}-1$
- float
    - 32 bits
    - -3.403E+38 to 3.403E+38
- double
    - Rational numbers
- char
    - Single characters
- String (not included)
    - Sequence of characters
        - Multiple letters or numbers in a row

Variable

- Storage of unknown amount
- Declared before used
    - Integer variable with identifier i

      int i ;
        - Default value will be 0
        - Integer variable with identifier i equal to 10

int i = 10;
        - Can set to equal after the fact
          int i ;
          i = 10 ;
    - double variable with identifier x
          double x;
        -
- Name is an identifier
    - Only 0-9, a-z, _ and $
    - Can't start with a digit
    - Are case sensitive
- Conventions
    - Variables and methods
        - First character is alphabetic and lowercase
    - Classes and interfaces
        - First character is alphabetic and uppercase
    - All are camelCase, meaning subsequent words are uppercase

Overflow

- When add 100 to maximum int value 2,147,483,647
    - Get -number
    - Loops back around
- Can also underflow
    - Subtract from smallest number and get very large number

Constant

- Constant never changes
  final static double
    - final means it can't be changed
    - Static saves memory
    - Double means it has a decimal

+ operator

- a++ ;
    - =a+1
    - First increases other stuff then increases x
    - ++a
        - First increases x then does other stuff

      a = 1

      x = a++ + 2;
      a = 2 x = 3

      x = ++a + 2;
      a = 2 x = 4

- a+= 1.6 ;
    - a = a + 1.6

- operator
- a--;
    - a = a-1
- a -= 10;
    - a = 10-1

/ operator
- a /=2;
    - a = a/2

* operator
- a *=10;
    - a = a*10

Int and double
- Int can be converted to double
- Double can't be converted to integer
    - Int = int/double
        - Fails
    - Double = int/double
        - Works
- Casting
    - Double to integer, just throws away/ignores the fractional part
        - Isn't rounding
        - To round, add .5 to fix when casting
            - If above x.5, then will get somewhere between y and y.5, where y=x+1, effectively rounding up
            - If below x.5, will get somewhere between x.5 and y where y=x+1, but will still cut off the fraction, rounding down
    - int a = (int) 2.95;
        - int a = 2
    -

Character

c = 'a';
- Need ' ' to show that its the character, not another variable
- ASCII is used for encoding ASCII values

  '0' = 48

  'A' = 65

  'a' = 97
- char one = 'a' + 1;
    - Output is b because of ASCII
        - Go from 97 to 98

Boolean
- True or false

- For logic, conditionals, and loops
- Default is set to false
    - Same as the rest in everything else

String

- Declared with “”
  String word = “hello world”;
- Defaults to null

Scanner

- Import it first
  import java.util.Scanner;
    - Must go before class
  Scanner *name* = new Scanner(System.in);
- Use method nextInt()
  int years;
  years = *name*.nextInt() ;
- Use method nextDouble()
  double distance;
  double = *name*.nextDouble();
- Use method next() (gets string)
  String personName;
  personName = *name*.next();
- Use method nextLine() (gets multi word string, including spaces and newline)
  String personFullName;
  personFullName = *name*.nextLine();
    - Faulty when use another scanner method before it
    - Includes newline character after each one
    - To fix add in a dummy nextLine that reads the newline character before your actual, functional, important nextLine

Method

- Take input and give output
- Always requires identifier followed by parenthesis
  Double value = Math.sin(2.4) ;
    - Method find sin of angle in radians in math class
    - Type is double
- Result = class.method.input
    - Math.abs()
        - Finds absolute value of input
    - Math.pow()
        - Returns first number to the power of the second number
    - sqrt()
        - Returns squareroot of input
    - Math.random()

        - Returns positive double between 0 and 1
        - [0.0, 1.0)
- Method signature is method name, and the number, type, order of parameters
- Static methods perform simple or specific task
    - Not connected to a particular object
- Scope
    - Variables from outside a method are not available inside a method
        - Only values can be passed in to the methods formal parameters
        -

Declare static method

- Start with always “public static”
- Next is the return type (int, double, String, void, etc.)
- Next is name of method
- Last is, in parentheses, formal parameters
- Inside curly braces put body of method
- Must include return statement within body of the method

Strings:

- Not primitive data type
    - Composed of a series of individual characters
- Initializing
    - String *name* = “”;
    - String *name* = new String(“”);
    - After initialized, is immutable
        - Can’t change the string
    - Reference a string method
        - stringname.methodname(arg 1, arg 2);
        - length() returns number of characters in a string including whitespace
            - First index is 0, so add 1 to get actual length
        - substring()
            - Returns a section of the string
            - Arg is which number of the string should be returned
                - If 6, then character 6 onwards
        - charAt()
            - Returns character at certain index
                - If arg is 6, then returns character 6 nothing else
        - indexOf()
            - Returns index of character
            - Inverse of charAt
                - If arg is “o” finds first index where “o” is used
                - If arg “o” for “Hello” would be 5
            - If character isn’t present in string, return value of -1
        - lastIndexOf()
            - Returns last index of character

                - Tells us last instance that character in arg is found
                - If character isn't present in string, return value of -1
            - compareTo(String anotherString)
                - Returns difference in strings based on ASCII values
                - If word 1 has higher ascii values, result is positive, if equal zero, if less negative
            - equals()
                - Boolean true if same, false if different
            - ==
                - Compares reference variable
                - Boolean true if comparing to reference variable, false if not
- Is a sequence of char
    - Reference variable is number in a box that then references the where the location of the sequences of boxes that is a string
        - A string is not a box containing a single value, but a series of boxes each containing a single char in a specific order
        - A reference variable is a single box that tells us where the stack of boxes aka string is.
            - Because immutable, can't change string, but can assign new string to same reference variable, achieving the same result

---

Java Types

- Primitive
    - Int
    - Double
    - Boolean
    - Char
    - Fixed amount of storage
        - Box to hold data
    - Use operators
- Object
    - String
    - Arrays
    - Infinite number of others
    - Hold arbitrarily complex data of any kind
        - No pre defined amount of storage
    - Use methods for operators
        - Consist of data and methods
- Primitive variable holds a value
- Reference variable tells us where the object is stored

  Turtle *refrenceVariable* = new Turtle();
    - constructor is method with same name as class
        - constructor signature is shorter than most methods

          Public name(type arg1, type arg2){

Boolean:

- Primitive type
    - Stores true or false
- Operator
    - ! not
    - && gives true if both are true, but false if anything else
        - Both have to be true to get true
    - || if either or both are true
        - Both have to be false to be false
    - ^ if either are true but not if both are true
        - Only one can be true to be true. Both false, is false. If both are true, is false
    - Order of evaluation is ! ^ && ||
- Algebra
    - Annulment law
        - AND with a 0 equals 0 or OR will equal 1
            - A && 0 = 0 always
            - A || 1 = 1
            - Doesn't matter what A is
    - Identity law
        - Term OR with 0 equals variable
        - Term AND with 1 always equals variable
            - A && 1 = A always
            - A || 0 = A
    - Idempotent law
        - A || A = A
        - A && A = A
    - Complement law
        - A && !A = 0
        - A || !A = 1
    - Absorption law
        - A || (A && B) = A
        - A && (A || B) = A
    - Distributive law
        - A && (B || C) = A && B || A && C
    - DeMorgan's law
        - !(A && B) = !A || !B
        - !(A || B) = !A && !B

- Statements
    - If

```
if(condition){
    statement;
}
```

        - Condition = true, then carries out
        - Condition = false, then doesn't
    - Else
        - For when two exclusive paths
            - Either are on time or late, if on time, else late
        - When if fails, goes to else
    - Else if
        - ONLY first true condition is executed
    - Switch

```
Switch (data){
    case value1:
        statement1;
        break;
    case value2:
        statement2;
        break;
    default:
        default statements;
```

        - break; stops the individual statements

Loops

- While loop
    - If statement where it repeats if condition is is still true

```
while (condition){
    Code stuff;
}
```

        - Sentinel value
            - Value that can't fit into set of data so stop code
- For loop
    - While loop with an int incrementing
    - Loop in variance
    - 3 parts in header
        - Initialization
          Int i = 0
        - Condition
          i < 302p
        - Update
          I++

- For each
    - Used for arrays
      for(type var: array)
- Do while
    - While loop that runs atleast once
      do{

      } while();
- Break
    - Keyword for loops that ends loop
- Continue
    - Keyword for loops that starts next loop

Array

- List of data of same type
    - Instantiated with type
      type name[ ];
      type[ ] name;
- Size is fixed after created
    - Ten elements would be as follows
      type[ ] name = new type[size of array];
- To find array elements
      type newname = name[point in array];
- Has .length; to find length of array
    - Different from .length();
- Assign multiple things at once
      type[] name = {arg 1, arg 2, arg 3…};
- Can also be:
      type[] name = new type[size];
      for(int i = 0; i < name.length; i++){
          name[i] = arg i;
      }
- For each loop
    - Takes each element of an array and changes it
      for(type name : array){
      Statements
      }
    - For loop that repeats for every element
    - Name = array at point in loop
- .toCharArray() turns String into array
    - Import java.util.Arrays
- Arrays.sort(array name) makes the array in order of smallest to largest
- Arrays.toString(array name) returns string

Files

- Use files need two statements
  ```
  import java.util.Scanner;
  import java.io.*;
  ```
- To read keyboard input we use scanner
- To use file, create scanner as well
  ```
  Scanner in = new Scanner(new File('in.txt"));
  ```
  - In.txt must be from same directory
  - Can be any file within ""

To check for exceptions, use try/catch

- try{
  all the code
  }
  catch(Exception io){
  What to do if there is an exception
  }

Another way is throwing the exception

- In method declaration add "throws FileNotFoundException"

  ```
  public static void main(String[] args) throws FileNotFoundException{
  }
  ```

Files again

- Uses same scanner methods
- Uses additional ones to check
  - hasNext() is boolean for if it has possibility for .next()
- Close scanner once done
  ```
  nameOfScanner.close()
  ```
- Delimiter breaks what is read into pieces
  ```
  scannerName.useDelimiter("[list of symbols to use as separators]");
  ```

Arrays

- 2d is an array of arrays aka table
  int[][]
- [rows][columns]
  Int[][] = {{row0}, {row1}, {nth row}}
- For each loop
  ```
  for(int [] temp : originalArray){
  }
  ```
- .length checks rows
  - Array[i].length checks columns for row i
  - Arrays.toString(name)
    - Turns array into string

Reference

- Reference = location in memory of specific object
    - Object = actual instance of class
- Compiler can recognize multiple instances of same string

```
String one = “Hello!”;
String two = “Hello!”:
System.out.println(one == two);
System.out. println(one.equals(two));

Prints true true
```

```
String one = "Hello!";
String two = new String("Hello!");
System.out.println(one == two);
System.out.println(one.equals(two));

Prints false true
```

- Java checks primitive values by value
- Method specific variables are separate from others
- Unless a class has own equals method, the class uses default, which only checks reference variables like ==
    - Like string
        - Works for content

ArrayLists

- Come in java util package

```
Import java.util.*;
```

- Can’t store primitives, only objects
    - There are wrapper classes that turn primitives into objects
        - Integer(int value)
        - Double(double value)
        - String toString()
        - .equals()

```
ArrayList<type> name;
```

- Type parameter or generic class
- Defaults to null
    - To instantiate, use

```
ArrayList<type> name = new ArrayList<type>();
```

- Methods:

`add(value)`	appends value at end of list
`add(index, value)`	inserts given value just before the given index, shifting subsequent values to the right
`clear()`	removes all elements of the list
`equals(list)`	returns true if two lists have the same size, contents, and order of elements.
`indexOf(value)`	returns first index where given value is found[1] in list (-1 if not found). [1] found means objects **equals(value)** method returned true. indexOf()
`get(index)`	returns the value at given index
`remove(index)`	removes & returns value at given index, shifting subsequent values to the left
`set(index, value)`	replaces value at given index with given value
`size()`	returns the number of elements in list
`toString()`	returns a string representation of the list such as `"[3, 42, -7, 15]"`

`addAll(list)` `addAll(index, list)`	adds all elements from the given list to this list (at the end of the list, or inserts them at the given index)
`contains(value)`	returns true if given value is found somewhere in this list. [1] found means object's **equals(value)** method returned true. contains()
`containsAll(list)`	returns true if this list contains every element from given list
`iterator()` `listIterator()`	returns an object used to examine the contents of the list (seen later)
`lastIndexOf(value)`	returns last index value is found in list (-1 if not found)
`remove(value)`	removes the first occurance of value from list. returns true if list contained the specified element
`removeAll(list)`	removes any elements found in the given list from this list
`retainAll(list)`	removes any elements *not* found in given list from this list
`subList(from, to)`	returns *a view* of the sub-portion of the list between indexes **from** (inclusive) and **to** (exclusive). more info
`toArray()`	returns the elements in this list as an array

- Can autobox
    - Put in primitive that it automatically converts to object
- To differentiate between .remove(value) and .remove(index), in .remove(value) cast it before the value to indicate
    - .remove((type) value)
- Can use for each loops with it
- To use .sort need import java.util.Collections
    - Collections.method()
        - .sort sorts it, greatest to least
        - .binarySearch needs to be in sorted list first, but will return index if is in list, if not in list, returns -index of where item would be
        - .rotate rotates it over by int
        - .reverse will flip it around and reverse it

Inheritance

- Parent class shares info with child class
    - Derived class inherits from base class
    - Child class can have only one parent class, but one parent class can have multiple child classes

- Two things not inherited
    - Constructors
    - Private instance variables
- Use keyword extends
    public class Circle extends Shape {
    Public class child extends parent
- constructor of child class has to have constructor of parent, and it must come first
    - Use super(arg 1, arg 2…)
    - If don't, autocalls default of parent
- Give default if parent has no constructor, if has initializing provides no auto default, so calling super() returns error
- this.instancevariable calls instance variable in current object

Polymorphism
- Overloaded method has same method name as another method in same class, but has different argument list
    - So different method signature
    - Like default constructor vs initializing constructor
- Confusing stuff idrk

Composition and Interfaces
- Composition
    - Relationship between class and other objects
    - Has-a relationship compared to inheritance which has is-a relationship
        - Vehicle and chassis/wheel/body/tire/hubcap
    - Put object in other
        - Shape has a point, so put point object within shape object
- Interface
    - Java interface is like class, but
        - Doesn't contain implementation of methods, only signature
        - Contains only method signatures and fields
        - Implements method with default after public/private
    - Framework that can then be overridden
        - Have capability or feature expressed that multiple want
        - Like interface flying object has fly and isflying then classes with interface would be bird, airplane, witch
    - If use interface, class must implement all methods
        - Each method declaration must match interface in method signature and return type
    - Use implements keyword to put it in just like extends for inheritance
    - All methods and objects in interface are automatically public static final
    - Can implement multiple interfaces
    Public class circle implements area, comparable, points {

Abstract Classes

- Basis for different subclasses that share behavior
    - Designed to be parents to several related classes with differing implementations
- Never instantiated but have constructors
    - Still need super

  Public abstract class Shape{
  Public class Circle extends Shape{
- Subclass must implement all methods in abstract classes
- Interface is contract that says you must give all abstract methods with identical method signatures
- Abstract methods are ideas of what is to be done
- Interface has all methods as abstract
    - Abstract classes can have some abstract methods
- Can extend only one abstract class
- Overridden method must have same method signature and return type

---

3, 11, 16, 20, 26, 29, 36, 41, 42, 46

Searches

- For loop to find in 1d array
- 2d array

```
Int count = 0;
for(int r = 0; r < arr.length; r++){
        for(int c = 0; c < (arr[r].length; c++){
                if(key == arr[r][c]){ count++; }
        }
}
```

- Worst case is search every one
- Binary search

```
Int low = 0;
Int high = arr.length - 1;
while(low <= high){
        Int mid = (high+low)/2;
        if(key.compareTo(arr[mid]) == 0){
                Return mid;
        }
        Else if(key.compareTo(arr[mid]) < 0){
                high = mid-1;
        }
        Else{
                Low = mid+1;
        }
}
Return -1;
```

- Worst case search is look at $-\log_2 N + 1$
- Java.util.Arrays has .sort() for arrays
- Java.util.Collections has .sort() for arraylists

Sort

- Bubble sort
  - Very unoptimized
    dbca
    bdca check if bd is in order
    Bcda check if dc is in order
    Bcad check if da is in order

    Loop
    Bcad
    Bacd

    Loop
    Abcd

    Size = #of elements in list
    for(i = 0 to size-2)
    Swap = false
    for(j = 0 to size-2 do
        If list[j] > list[j+1]
            Swap

- Selection sort
  - Find smallest, set it to index 0
  - Find second smallest, set it to index 1
  - Find third smallest index, set it to index 2
  - Keep on going
-

Recursion

- When a method calls itself