Interpreted languages: requires an “interpreter” to run the code, code is checked for errors as it runs(Python, JS)
Compiled languages: requires an “compiler” to run the code, code must be compiled and free of error before running(Java, C)
Procedural languages: creates functions & variables that are separate from the program, can work together but its separated
Object - oriented languages: the concept that object’s functions and variables are encapsulated together so its all connected
Java: requires Java Development Kit(JDK) & Java integrated Development Environment(IDE)
Object: actual thing being built from the blue print
Instance: a virtual copy of the object
Java program has two files: source code file(fileName.java developer created) + executable file(fileName.class auto created)

Class header
- syntax: accessModifier class className
- Example: public class studentPopulation

Main method: public static void main(String [] args)
System.out.print

Comments: single line //
Multi-line /* */

Access modifiers:
- Public class: can be used by any other class
- Private class: can only be used within the same package

Name conventions:
- Start class with upper case letter
- Capitalize each word (ex: studentSchoolPopulation)

Programming errors:
- Syntax error: detected by compiler
- Runtime error: causes the program to abort
- Logic error: produces incorrect result

Data types:
- Primitive data: byte, short, int, long, float, double, char, boolean
- Object: java has many types of objects, and can invent as much others as needed

Scanner: used to input a value from the user/keyboard
```
import java.util.Scanner;
Scanner userInput = new Scanner(System.in);
String name = userInput.nextLine();
```
Object is defined by:
- characteristics(Attributes)
- Actions(Methods/Functions)

Class is defined by:
- Instance variables: describes object’s characteristics
- Methods: specify actions that can be performed by the object
- Types of classes:
    - defined by java library: String, Scanner, System
    - defined by user: Car, School, Boolean

Variables:
- Instance variables: variables declared inside a class but outside of method bodies
- Local variables: variables declared in the body of methods, lost when method terminates

Instantiating an instance of a class: create new object of a class by new keyword
- Example: Student student1 = new Student(“Cindy”, 20);

Constructor: method called to initialize instance variables of the class
No-argument constructor: initializes instance variables with default values
Private
- Private methods: can only be used in the methods of the class in which its declared
- Private classes: can only be used in classes in the same package

This.keyboard: uses whenever to refer to the current object’s instance variable
Static: only one of it can exists in the entire run of the program, belongs to the class, not the individual object
UML(Unified Modeling Language): graphical scheme for modeling object-oriented systems, used to communicate with programmers
- Access modifiers:
- public : +
- Private: -
- Variables:
- Syntax: <accessModifier><variableName> : <datatype>
- Example: +name:String explanation: public String name;
- Example: +balance:double   explanation: public double balance;

- Methods:
- Syntax: <accessModifier><methodName>(parameters) : <outputDatatype>
- Example: +setBalance(balance:double)
- explanation:public double setBalance(double balance){
Return balance; }
- Example: -getArea():double
- Explanation: private double getArea(){
Return area; }

Class Structure:

```
        Public class Car{
                Private int numDoors;
                Private int speed;
                Private String color;

                Public void run(){
                }
```

Declaration: String name;
Initialization: name = “Cindy”;
Constants: values that isn’t going to change during programming execution, best practice: capitalize the variable
- Example: final double TAX = 0.25;

If statement syntax:

```
        If (condition) {
           //       Statement block #1
        } else {
         //         Statement block #2
        }
```

Multi-branch If statement syntax:

```
        If (condition){
          //        Statement block #1
        } else if(condition#2){
          //        Statement block #2
        } else if(condition#3){
          //        Statement block #
        }
```

Logical operators:
- OR || : evaluates to true of <u>one</u> of the components is correct
- AND && : evaluates to true if <u>all</u> components are correct
- NOT ! : converts non-zero operand into 0, and a zero operand in 1

While loop statement:
- Checks condition <u>before</u> first execution of the loop
- Loop body isn’t executed if condition if false
- Loop body can be executed <u>zero</u> or more times

```
         Syntax: while (condition) {
                        Block of code
                 }
```

Do-while loop:
- Checks the condition <u>after</u> the first execution of the loop
- Loop body can be executed <u>one</u> or more times
- Loop body is guaranteed to run 1x

```
                Syntax: Do{
                                // code to execute
                        } while (condition);
```

Counting loop: number of iterations is fixed and specified before loop starts
Sentinel-controlled loop: number of iterations depends on user input
Result-controlled loop: number of iterations depends on result of calculation performed by the loop’s statements

Static method: - Static method: belong to the class
- without-static: belong to the instance of the class

Encapsulation: make sure that "sensitive" data is hidden from users, by private variable/methods. Wrapping variables & functions together
Inheritance: child class can use methods from the superclass, using extend keyword
Polymorphism: many forms, use the same method differently
Abstraction: hiding certain details and only showing essential info
Interface: methods with empty bodies, used with implemented keyword