Java Notes Chapter 04 – More Object Concepts

**UNDERSTANDING BLOCKS & SCOPE:**

\*\*In Java: a variable’s scope level is its block

\*\*A locally declared variable always masks/overrides another variable with the same name somewhere else in the class ( ex: a class’s instance fields / variables )

* Overloaded Method:
* Multiple methods share a name; the compiler understands which one to use based on the arguments in the method call. (method signature)
* Methods created to accept different argument types

\*\*Java can promote one data type to another when you pass parameters(s)

* Ex. Method has a double parameter 🡪 can pass an int
* You can call methods using the correct type or a type that can be promoted

\*\*Overloading can lead to **ambiguous** situations: (method doesn’t know which one to use)

* Java makes no assumptions based on variable names & doesn’t consider the return type

Constructors with parameters: often used to initialize data fields for an object

* Overloaded Constructor:
* aka constructor w/ parameters
* Allows for creation of objects w/ initializing arguments, or none as needed

( aka constructors with or without parameters )

* **this** Reference: refers to the class’s field variables
* Allows all objects to use just one copy of each non-static method
* Automatically receives memory address of the object it references
* Only non-static, instance methods have a ‘this’ reference
* ‘this’ makes overloaded constructors more efficient:
* When each constructor contains similar statements, call one constructor version from another

Static: ( using static fields )

* Methods: have no ‘this’ reference bc there’s no objects associated with them
* Static methods 🡪 Class Methods
* Static variables 🡪 Class Variables

\*\*Static Methods cannot access instance variables

\*\*Non-Static (instance) Methods can access both static & instance variables

Final: ( using constant fields )

* Constant: is a final field / variable
* Static: all objects share a single memory location for the field
* Final: cannot change during the program

\*\*Non-Static final field’s value can be assigned in a constructor

\*\*Static final field’s value must be set at declaration

**AUTOMATICALLY IMPORTED / PREWRITTEN:**

* Java has 2 categories of packages:
* java.lang: imported into every program & contains fundamental classes
* All other packages are only available if you import them (optional classes)

Ex. Import javax.swing package 🡪 JOptionPane

* **Math** Class: java.lang.Math
* Contains constants & methods for common math functions
* All are Class Variables & Methods (static)

\*\*\*You don’t / can’t instantiate objects of type Math – the constructor is private

\*\*Ex. PI:

\*Commonly used Math constant 🡪 public final static double PI = 3.14159..

- public: so any program can access it directly

- final: so it cannot be changed

- static: so only one copy exists & you can access it without declaring a Math object

- double: so it holds a floating-point value

* Using classes not automatically imported:
* Use entire path with class name 🡪 java.time.LocalDate myAnniversary ;
* Import the class 🡪 ex. Import.java.util.Scanner;
* Import entire package - using ‘ \* ’ in import statements:
* Imports all classes in a package (but not other packages w/in that package)
* LocalDate Class:
* now(), of() -Static Methods (used instead of the new operator)

\*you know they’re static bc they’re used w/ class name & not an object

* getMonth(), getDayOfWeek() - return enumerations

**COMPOSITION:**

* The relationship between classes when an object of one class is a data field within another

\*\*When you use an object as a data member of another object, you must remember to supply values for the contained object if It has no default constructor

**NESTED CLASSES:**

* In Java you can create a class within another class and store them together
* Containing Class 🡪 top-level class. 4 types of nested classes
* Static Member: has access to all static methods of top-level class
* Non-Static Member: requires an instance; has access to all data & methods in top-level
* Local: local to a block of code
* Anonymous: local classes with no identifier

\*\*Most common reason to nest a class inside another is bc the inner class is used only by the top-level class