JAVA

General

- A computer's hardware + software
- The core of a computer has memory (stores data) and CPU (executes programs)
- A computer has two types of memory **main memory** and **secondary memory**
- Main memory is **volatile**, meaning the stored information is maintained only as long as the computer is on.
- Secondary memory is **non-volatile** and retains stored information even when the computer is turned off.
- Computers connected to share information are called a network.
- A network can be wired or wireless as well as **local** (computers in small areas) or **wide** (computers in a large area)
- The **internet** is a wide area network
- Computers communicate using **protocols**
- Computers on the internet must use **TCP/IP protocols** to communicate
- CPUs can only execute **binary**
- Data on a computer is also stored in binary (whether it: SSDs (Solid State Drives), HDDs (Hard Disk Drives), and RAM)
- The **fetch-decode-execute** cycle is the basic process of a CPU.
- The CPU has an **arithmetic/logic unit** for performing logic and calculations.
- **Registers** in the CPU provide limited storage for temporary values during calculations.

- A **program** is a series of coded instructions for a CPU to execute
- Binary is represented in 0 and 1
- Each digit in the binary is called a "bit."
- A ***byte*** is made of 8 bits. A ***kilobyte*** is 1024 bytes. A ***megabyte*** is 1024 kilobytes. A ***gigabyte*** is 1024 megabytes.
- We can't easily code in binary so we create human-readable **languages** that are later translated to binary.
- A programming language sets rules for combining words and symbols into executable program instructions.
- Programming languages are translated to binary in either or a hybrid of two ways: **compilation** or **interpretation**.
- ***Compiler*** (compilation) - translates your entire code first before it begins to be executed
- ***Interpreter*** (interpretation) - translates code to binary line by line while it executes it
- Languages come in ***4 levels of abstraction*** (how far from binary they are):
  - **Machine** binary -
  - **Assembly** -
  - **High level** - Human-readable syntax (e.g., Java, Python, C++)
  - **Fourth generation** - Often using declarative statements to automate complex tasks

Program Structure

- **Java** is a programing language
- Java code is first compiled to **bytecode** instead of binary
- Java ***bytecode*** is a low-level representation of a Java source code program

- A Java interpreter (translates to binary and executes line by line) called the **Java Virtual Machine** (JVM) executes the Java bytecode.
- Java is an object oriented programming language which means programs are structured as interacting objects rather than a sequence of instructions. These objects are defined with classes.
- An object is defined by a **class**. A class is the model or blueprint from which an object is created.
- An object has **state** (descriptive characteristics) and **behaviors** (what it can do or what can be done to it)
- A class defines an object's state with **instance variables** and its behavior with **methods**.
- A program is made up of one or more classes
- The class that contains the **main method** of a Java program represents the launch/start method of the program (also called the **driver class**)
- A Java application always contains a method called main (the launch/start method of the entire program)

Comments

- Single-line comments (//) are for short explanations or disabling a single line of code.
- Line comments (/* ... */) are for longer descriptions spanning multiple lines.
- Javadoc comments (/** ... */) Used for documenting classes, methods, and fields in Javadoc format.
- Java doc always starts with a description on the first line.
- Javadoc comment has @author for defining the author (used mostly on classes)
- @param [parameter name] is used describe a method/ constructor parameter

- @return [] what a method is returning
- Javadoc comments can have HTML tags inside them

Identifiers

- Identifiers are the "words" in a program
- A Java identifier can be made up of letters, digits, the underscore character (_), and the dollar sign
- Identifiers cannot begin with a digit
- Java is case sensitive: Total, total, and TOTAL are different identifiers
- Reserved words : are predefined words in the Java language that have a **specific meaning** and **cannot be used as identifiers**
- Reserved words: abstract, assert, boolean, break, byte, case, catch, char, class, const, continue, default, do, double, else, enum, extends, false, final, finally, float, for, goto, if, implements, import, instanceof, int, interface, long, native, new, null, package, private, protected, public, return, short, static, strictfp, super, switch, synchronized, this, throw, throws, transient, true, try, void, volatile, while.

Errors

- A program can have three types of errors
- Compile-time Errors - happen before code runs, an executable version of the program is
  not created
- Runtime Errors - happen while the program is running, eg. trying to divide by zero
- Logical Errors - A program may run, but produce incorrect results, perhaps using an incorrect formula

Problem Solving

- These steps can overlap:
  - Understand the problem
  - Design a solution
  - Consider alternatives and refine the solution
  - Implement the solution
  - Test the solution

Classes & Objects

- An object is defined by a **class**. A class is the model or blueprint from which an object is created.
- An object has **state** (descriptive characteristics) and **behaviors** (what it can do or what can be done to it)
- A class defines an object's state with **instance variables** and its behavior with **methods**.
- An object is an instance of a class that contains state (fields/variables) and behavior (methods)
- A program is made up of one or more classes

Data Expressions

Variables:

- A variable is a name for a location in memory used to hold a data value
- The structure for defining a variable is as follows : <accessModifier?> <non-accessModifier?> <dataType> <variableName> = <value>;
- AccessModifier → Defines the visibility of the variable (public, private, protected). Its optional.
- non-accessModifier - Optional. Defines the mutability of the variable (e.g. final)

- DataType → Specifies the type of data the variable can hold.
- VariableName → The unique identifier of the variable.
- "=" (Assignment Operator) → Assigns a value (optional at declaration).
- value → The actual data stored in the variable (optional at declaration).
- In Java, a variable goes through different stages in its lifecycle:
  - Declaration – Defining the variable with a data type.
  - Initialization – Assigning an initial value when declaring the variable.
  - Assignment – Giving a value to a previously declared variable.
- Depending on the access modifier or data type of a variable, it can be mutable or immutable.
- A variable's **mutability** refers to whether its value can be **changed after initialization**.
- A **mutable variable** is one whose value **can be changed** after it is assigned.
- An **immutable variable** is one whose value **cannot be changed** after it is assigned. (they can only be declared and initialized)
- You can use the non-access modifier 'final' to declare any variable as immutable.
- Two variables are aliases of each other when they refer to the same object.

Primitive Data Types

- A data type specifies what kind of data a variable can store and what operations can be performed on it.
- In Java, data types are categorized into primitive types and reference types.
- Primitive data are basic values such as numbers or characters, not represented as object.
- Java has **8 primitive data types**, each with a specific size and purpose.
- `byte` **(8-bit)** – Stores small integers from -**128 to 127**.
- `short` **(16-bit)** – Stores integers from -**32,768 to 32,767**.

- `int` **(32-bit)** – The default integer type, storing values from -2^{31} **to** 2^{31}**-1**.
- `long` **(64-bit)** – Stores very large integers from- 2^{63} **to** 2^{63}**-1**.
- `float` **(32-bit)** – Stores **floating-point (decimal) numbers**, less precise than `double`.
- `double` **(64-bit)** – Stores **floating-point numbers** with **higher precision**.
- `char` **(16-bit)** – Stores a **single Unicode character** (e.g., `'A'`, `'@'`).
- `boolean` **(1-bit)** – Stores `true` or `false` values only.

Reference Data Types

- Reference types are objects and store the memory address (reference) of the object, not the actual value.
- **String** - Represents a sequence of characters and is immutable. When reassigning a new string object is created instead of the old one being modified.
- **Integer** - Wrapper class for `int`, allows null values and utility methods.
- **Double** - Wrapper class for `double`, provides precision with floating-point numbers.
- **Float** - Wrapper class for `float`, used for single-precision decimal numbers.
- **Long** - Wrapper class for `long`, used for very large whole numbers.
- **Short** - Wrapper class for `short`, a memory-efficient alternative to `int`.
- **Byte** - Wrapper class for `byte`, used for small numbers in low-memory applications.
- **Boolean** - Wrapper class for `boolean`, allows `null` and logical operations.
- **Character** - Wrapper class for `char`, provides methods for character operations.

Data Conversion:

- Data can be converted from one data type to another

- A conversion between one primitive type and another falls into one of two categories:
  widening conversions and **narrowing conversions.**
- Widening goes from one data type to another type that uses an equal or greater
  amount of space to store the value.
- Widening conversion go from one type to a type that uses less space to store a value.
- Narrowing conversions should be avoided because they can lose information.
- In Java , conversions can occur in three ways:
  - assignment conversion - Only when widening, its implicit
  - promotion - occurs automatically when certain operators
    need to modify their operands in order to perform the operation
  - casting - specify the type to convert to by name in parentheses

Expressions

- An expression is a combination of one or more operators and operands that usually perform a calculation.
- Increment operator (++) - used to increase a number value by 1.
- Decrement operator (- -) : used to decrease a number value by 1.
- Increment/Decrement operators can be written as ++count or count++.
- Assignment operator (+=/ -=) : subtract or add then assign.
- Addition (`+`) - Adds two values together. Works on: `int`, `long`, `float`, `double`, `byte`, `short`, `char` (auto promoted to int) , `String` (for concatenation)
- Subtraction (`-`) - Subtracts one value from another. Works on: `int`, `long`, `float`, `double`, `byte`, `short`, `char` (auto promoted to int) .
- Multiplication (`*`) - Multiplies two values. Works on: `int`, `long`, `float`, `double`, `byte`, `short`, `char` (auto promoted to int)

- Division (`/`)- Divides one value by another. Works on: `int` , `long` , `float` , `double` , `byte` , `short` , `char` .
  - Integer division (`int / int`) truncates decimals.
- **Remainder (`%`)**- Returns the remainder of division (modulus). **Works on:** `int` , `long` , `float` , `double` , `byte` , `short` , `char` .
  - Modulus will be negative if the numerator (*numerator* % x) is negative
  - Modulus will be zero if x divides into the numerator fully.
- Java follows PEMDAS when evaluating the order of arithmetic operators.
- A Boolean expression is an expression that evaluates to a true or false result
- `!` **(Logical NOT)** - Reverses a boolean value (`true` becomes `false` , `false` becomes `true`).
- `&&` **(Logical AND)** - Returns `true` only if **both** conditions are `true` , otherwise returns `false` .
- `||` **(Logical OR)** - Returns `true` if **at least one** condition is `true` , otherwise returns `false` .
- `==` **(Equal to)** - Returns `true` if two values are **equal**, otherwise `false` .
- `!=` **(Not equal to)** - Returns `true` if two values are **not equal**, otherwise `false` .
- `>` **(Greater than)** - Returns `true` if the left value is greater than the right value.
- `<` **(Less than)** - Returns `true` if the left value is less than the right value.
- `>=` **(Greater than or equal to)** - Returns `true` if the left value is greater than or equal to the right value.
- `<=` **(Less than or equal to)** - Returns `true` if the left value is less than or equal to the right value.
- Short circuiting is when a boolean expression does not have to be completely evaluated
  - && is short circuited when the first of two terms in the expression evaluates to false rendering the whole expression as false.

    - || is short circuited when the first of two terms in the expression evaluates to true rendering the rest of the expression as true.
- For comparing chars and strings you must use compareTo(), equals(), and equalsIgnoreCase() methods.

Reading User Input

- The **Scanner** class is part of the `java.util` package and must be **imported** to be used.
- The `nextLine()` method reads all of the input until the end of the line is found.
- The `nextInt()` method reads an integer input from the user.
- The `nextDouble()` method reads a floating-point number from the user.
- The `next()` method reads a single word (stops at whitespace).
- The **Scanner** class can be used to read **input from the keyboard, files, or other input streams**.
- The `useDelimiter()` method allows changing the default input separator (whitespace).
- The **Scanner object** should be closed using `close()` to release system resources when done.

User Output

- `System.out.print()` - Prints text **without** moving to a new line.
- `System.out.println()` - Prints text and **moves to a new line**.
- `System.out.printf()` - Formats output using placeholders (`%d`, `%f`, `%s`).
- **Decimal formatting (`DecimalFormat` from `java.text.DecimalFormat`)**
    - **Must be imported:** `import java.text.DecimalFormat;`

- **Formats decimal numbers to a specific pattern.**
- **Example usage:**

```
DecimalFormat df = new DecimalFormat("#.##");
System.out.println(df.format(12.3456)); // Output: 12.35
```

- **General number formatting (`NumberFormat` from `java.text.NumberFormat`)**
  - **Must be imported:** `import java.text.NumberFormat;`
  - **Used for locale-specific formatting like currency or percentages.**
  - **Example usage:**

```
NumberFormat currency = NumberFormat.getCurrencyInstance();
System.out.println(currency.format(1234.5)); // Output: $1,234.50 (dep
ending on locale)
```

Creating Objects

- **A class** is declared using the `class` keyword and serves as a blueprint for creating objects.
- **Instance variables** (fields) are declared within a class but outside methods, and they hold data specific to each object.
- **Local variables** are declared inside methods or blocks and exist only within that scope.
- **Constructors** are special methods used to initialize objects and have the same name as the class.
- **Objects are created using the `new` keyword**, which invokes the class constructor.

- **Access modifiers (`public`, `private`, `protected`, package-private)** control how class members (variables and methods) are accessed.
  - A `public` variable, method, or class **can be accessed from anywhere** in the program
  - A `private` variable or method **can only be accessed within the same class**.
  - A protected variable or method can be accessed within the same package and by subclasses (even if they are in different packages).
  -
- **The `static` keyword** makes a variable or method belong to the class itself rather than to instances of the class.
  - Memory space for an instance variable is created for each object that is instantiated from a class. A static variable is shared among all objects of a class.
- **Multiple objects of the same class** can be created, each with its own set of instance variables.
- **If no constructor is defined, Java provides a default constructor** with no parameters.

Methods:

- **A method** is a block of code that performs a specific task and can be called multiple times.
- **Methods are defined inside a class** and contain a return type, name, parentheses, and a body.
- **The return type** specifies the type of value the method returns (`void` if it does not return anything).
- **Parameters** allow methods to receive input values when called.
- **Method overloading** allows multiple methods in the same class to have the same name but different parameter lists.

- **Method arguments** can be passed **by value** (primitives) or **by reference** (objects).
- **Access modifiers (`public`, `private`, `protected`)** determine method visibility.
- `static` **methods** belong to the class and can be called without creating an object.
- `final` **methods** cannot be overridden by subclasses.
- **Recursion** is when a method calls itself to solve a problem.
- **A method must be called (invoked)** for its code to execute.
- **Getter methods (`getX()`)** retrieve instance variable values, while **setter methods (`setX()`)** modify them.

Predefined Classes

Math Class (`java.lang.Math`)

- The Math class is part of `java.lang` and does not need to be imported.
- **It provides mathematical operations and constants (`PI`, `E`).**
- **Methods are `static`, meaning they can be called without creating an object.**
- **Common methods include:**
  - `Math.abs(x)` - Returns the absolute value of `x`.
  - `Math.pow(x, y)` - Returns `x` raised to the power of `y`.
  - `Math.sqrt(x)` - Returns the square root of `x`.
  - `Math.max(a, b)` - Returns the larger of `a` and `b`.
  - `Math.min(a, b)` - Returns the smaller of `a` and `b`.
  - `Math.round(x)` - Rounds `x` to the nearest integer.
  - `Math.floor(x)` - Rounds `x` down to the nearest whole number.
  - `Math.ceil(x)` - Rounds `x` up to the nearest whole number.

- `Math.random()` - Returns a random number between `0.0` (inclusive) and `1.0` (exclusive).

Random Class (`java.util.Random`)

- The Random class is part of `java.util` and must be imported using `import java.util.Random;`.
- Used to generate pseudo-random numbers.
- Requires creating a `Random` object before use (`Random rand = new Random();`).
- Common methods include:
  - `nextInt()` - Returns a random integer.
  - `nextInt(n)` - Returns a random integer from `0` to `n - 1`.
  - `nextDouble()` - Returns a random `double` between `0.0` and `1.0`.
  - `nextFloat()` - Returns a random `float` between `0.0` and `1.0`.
  - `nextBoolean()` - Returns `true` or `false` randomly.

String Class (`java.lang.String`)

- **The String class is part of `java.lang` and does not need to be imported.**
- **A `String` is immutable (cannot be changed after creation).**
- **The `.length()` method returns the number of characters in a string.**
- **Common methods include:**
  - `charAt(index)` - Returns the character at the specified index.
  - `substring(start, end)` - Extracts a substring from a string.
  - `toUpperCase()` - Converts all letters to uppercase.
  - `toLowerCase()` - Converts all letters to lowercase.
  - `trim()` - Removes leading and trailing whitespace.
  - `replace(oldChar, newChar)` - Replaces all occurrences of a character.

- `equals(str)` - Compares two strings for exact equality.
- `equalsIgnoreCase(str)` - Compares two strings, ignoring case.
- `contains(sequence)` - Checks if a string contains a sequence of characters.
- `compareTo(otherString)` - compares a string lexicographically to another. Checks the Unicode of each character in the string and returns the first difference it finds.
- `startsWith(prefix)` - Checks if a string starts with a given prefix.
- `endsWith(suffix)` - Checks if a string ends with a given suffix.
- `split(delimiter)` - Splits a string into an array based on a delimiter.
- `indexOf(str)` - Returns the index of the first occurrence of a substring.
- `isEmpty()` - Returns `true` if the string is empty (`""`).

Enums (`enum`)

- An `enum` (enumeration) is a special class that represents a fixed set of constant values.
- Declared using the `enum` keyword.
- Enum constants are implicitly `public`, `static`, and `final`.
- Can contain methods, constructors, and instance variables.
- Useful for defining a set of predefined values, like days of the week or status codes.
- Example declaration:

```
enum Day { SUNDAY, MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRI
DAY, SATURDAY }
```

Records (`record`)

- **A `record` is a special type introduced in Java 14 to reduce boilerplate code.**
- **Automatically generates constructors, getters, `equals()`, `hashCode()`, and `toString()`.**
- **Used for immutable data objects.**
- **Declared using the `record` keyword.**
- **Example declaration:**

```
record Person(String name, int age) {}
```

Autoboxing and Unboxing

- **Autoboxing** automatically converts a **primitive type** into its **corresponding wrapper class**.
- **Unboxing** automatically converts a **wrapper class** object back into a **primitive type**.
- **Eliminates the need for manual conversions between primitives and objects.**
- Works with `Integer`, `Double`, `Boolean`, `Character`, etc