**Mid-Term Exam Study Guide: Key Terms and Definitions**

**General Java Concepts**

- **Java Program Structure:** Every Java program must have at least one class, and the program's execution begins from the main method within that class.
- **IDE (Integrated Development Environment):** A software application that provides comprehensive facilities to computer programmers for software development, integrating editing, compiling, building, debugging, and online help within a single graphical user interface. Examples include JGRASP, NetBeans, and Eclipse.
- **Comments:** Lines of code ignored by the compiler, intended to enhance readability.
    - **Line Comment:** Starts with // and comments out the rest of the line.
    - **Block Comment:** Enclosed between /* and */ for one or several lines.
    - **Javadoc Comment:** Special type of comment beginning with /** and ending with */, used for commenting on an entire class or method, and can be extracted into an HTML file.
- **Import Statement:** Used to inform Java of your intention to use classes from a specific package, such as the Scanner class from java.util.
- **Scanner Class:** Used for reading input from the console. It needs to be imported from the java.util package. You typically need only one Scanner object per program, which can read different data types using methods like nextInt(), nextDouble(), nextLine(), etc..
- **Variables:** Used to store values that can be changed in the program. They must be declared by specifying their type before use.
- **Constants:** Special types of variables whose stored value is unchanged. By convention, constants are capitalized with underscores between words.
- **Assignment Statement:** Used to assign a value to a variable, typically using the single equal sign (=).
- **Primitive Data Types:** Eight fundamental data types in Java: byte, short, int, long (for integers), float, double (for floating-point numbers), boolean (for logical operations, holding true or false), and char (for characters).
- **Mathematical Expressions/Operators:** Include addition (+), subtraction (-), multiplication (*), division (/), and remainder (%).

    - **Integer Division:** When both operands are integers, the division operator (/) performs integer division, discarding any fractional part.
    - **Remainder Operator (%):** Returns the remainder of an integer division.
- **Literals:** Terms used to signify specific values, especially numeric values like 5, 10, or 3.14, or true/false for boolean values, and single characters.
- **Identifiers:** Names given to variables, methods, classes, and other program elements. They can use characters, digits, underscore (_), and dollar sign ($), but cannot start with a digit or be a reserved word. Java is case-sensitive for identifiers.
- **Compilation:** The process where a Java source file (.java) is translated into a Java bytecode file (.class) by a Java compiler (javac command).
- **Execution:** The process of running the compiled Java bytecode on the Java Virtual Machine (JVM).
- **Programming Errors:**
    - **Syntax Errors (Compile Errors):** Violations of Java's grammatical rules, reported by the compiler. Examples include missing semicolons, braces, or misspelled words.
    - **Runtime Errors:** Errors that occur while a program is running, causing it to terminate abnormally. Often caused by impossible operations or input mistakes.
    - **Logic Errors:** Occur when a program compiles and runs but does not perform the way it was intended, producing incorrect or unexpected results.
- **Software Development Process:** A logical way of writing programs, typically involving:
    - **Analysis:** Understanding the problem, identifying inputs, outputs, and requirements.
    - **Design:** Developing an algorithm (logical steps) to process inputs and produce outputs.
    - **Implementation (Coding):** Translating the algorithm into Java code.
    - **Testing:** Compiling, running, and verifying the code with sample data.
    - **Deployment:** Delivering the software for use.

    - **Maintenance:** Fixing issues reported by the client.

**Selections (Chapter 3)**

- **Selection Statements (Decision Structures):** Allow programs to deviate from sequential execution and choose different paths based on conditions.
- **Flow of Control:** The order in which program statements are executed. By default, it's sequential, but selection and loop structures can alter this flow.
- **Boolean Expression:** An expression that evaluates to either true or false.
- **Relational Operators:** Used to compare two values (<, <=, ==, !=, >, >=). The double equal sign (==) is for comparison, while a single equal sign (=) is for assignment.
- **Logical Operators:** Work on boolean variables/expressions (&& for AND, || for OR, ^ for XOR, ! for NOT).
- **Operator Precedence:** Rules that determine the order in which operators in an expression are evaluated. Mathematical operators generally have higher precedence than relational operators, which have higher precedence than logical operators. Parentheses always take the highest priority.
- **if Statement:** Executes a block of statements only if a condition is evaluated to true.
- **if-else Statement:** Allows choosing between two options: one set of actions if the condition is true, and another if it's false.
- **Conditional Operator (Ternary Operator):** A shortcut syntax for an if-else statement, taking three parameters: a condition, a value if true, and a value if false.
- **Nested if Statements:** An if statement placed inside another if or else block.
- **switch Statement:** Allows choosing from multiple options based on a discrete (countable) expression (e.g., integer or character types).
    - **break (in switch):** Terminates the execution of the switch statement, jumping to the next line after the switch block.
- **Semicolon After if:** Placing a semicolon immediately after an if statement's condition (if (condition);) creates a logical error, as it prematurely terminates the if statement, causing the associated block to always execute regardless of the condition.

**Math Methods, Characters, and Strings (Chapter 4)**

- **Math Class:** A built-in Java class containing a rich collection of mathematical functions (e.g., sin, cos, pow, sqrt, ceil, floor, random, min, max, abs) and constants (PI, E). It is a "static class" meaning its methods are called using the class name directly (e.g., Math.pow()). It is implicitly imported from the java.lang package.
- **char Data Type:** Represents a single character.
    - **Single Quote vs. Double Quote:** Single quotes (' ') are used for characters, while double quotes (" ") are used for strings.
    - **Unicode:** A universal coding scheme used to represent characters in Java. Characters can be initialized with their actual character or their Unicode value.
- **Escape Sequences:** Special characters preceded by a backslash (\) that represent control characters (e.g., \n for newline, \t for tab, \\ for backslash, \' for single quote, \" for double quote).
- **String:** A sequence of characters enclosed in double quotation marks. Strings are objects in Java, not primitive types, and are immutable (their value and length don't change once created).
    - **String Concatenation:** The process of joining two or more strings, typically using the + operator.
    - **Common String Methods:** length() (returns length), concat() (joins strings), toUpperCase()/toLowerCase() (changes case), replace() (replaces characters), substring() (extracts a portion), trim() (removes leading/trailing spaces), charAt() (gets character at an index), indexOf()/lastIndexOf() (finds character/substring index), equals()/equalsIgnoreCase() (compares strings), startsWith()/endsWith() (checks prefix/suffix).
- **printf Statement:** An alternative way to print formatted output to the console, allowing control over spacing, decimal places, and alignment using format specifiers.
    - **Format Specifiers:** Characters preceded by a percent sign (%) that indicate how data should be formatted (e.g., %b for boolean, %c for character, %d for decimal integer, %f for floating-point, %e for scientific notation, %s for string).

- **Type Conversion (Casting):** The process of storing a value of one data type into a variable of a different data type.
    - **Widening Conversion (Implicit):** Converting a smaller data type to a larger one (e.g., int to double). This is generally safe and happens automatically.
    - **Narrowing Conversion (Explicit):** Converting a larger data type to a smaller one (e.g., double to int). This may lead to loss of data and requires explicit casting using parentheses (e.g., (int)).

**Loops (Chapter 5)**

- **Loops (Repetition Statements):** Programming constructs that allow statements to be executed repeatedly.
- **while Loop:** A conditional loop that repeatedly executes its body as long as a boolean condition remains true. The condition is checked *before* each iteration, so the loop body may execute zero or more times.
- **do-while Loop:** Similar to a while loop, but the condition is checked *after* each iteration, guaranteeing that the loop body executes at least once.
- **for Loop:** A counting loop often used when the number of repetitions is known beforehand. It typically consists of three parts: initialization, loop-continuation condition, and increment/decrement.
- **Loop Body:** The block of statements that are executed repeatedly by a loop.
- **Conditional Loops (Nondeterministic Loops):** Loops (like while and do-while) where the number of repetitions is not known beforehand, as it depends on a condition.
- **Deterministic/Counting Loop:** A loop (like for) where the number of repetitions is known or predictable due to a counter.
- **Infinite Loop:** A loop that runs indefinitely because its termination condition is never met. Often caused by errors in the loop's condition or increment/decrement.
- **Sentinel Value:** A special input value used to control the termination of a loop, signaling that the user wants to stop providing input.
- **Nested Loops:** A loop placed inside the body of another loop. The inner loop completes all its iterations for each single iteration of the outer loop.

- **break Statement (in loops):** Terminates the *current* loop immediately, jumping control to the statement immediately following the loop.
- **continue Statement (in loops):** Skips the *current* iteration of the loop, moving control to the end of the loop body to start the next iteration.
- **Input Validation:** Using loops (e.g., while loops) to repeatedly prompt the user for input until a valid value is entered.

**Methods (Chapter 6)**

- **Method:** A collection of statements grouped together to perform a specific operation or task. Methods help in code reuse, modularity, and reducing complexity.
- **Method Header:** Defines the method's characteristics, including modifiers (public, static), return type (int, void, etc.), method name, and parameter list.
- **Method Body:** The block of code enclosed in braces {} that contains the detailed implementation for the method's task.
- **Formal Parameters (Parameters):** Variables declared in the method header that act as placeholders for values passed into the method when it's called. They are treated as local variables within the method.
- **Actual Parameters (Arguments):** The actual values or expressions passed to a method when it is invoked. These values are copied to the formal parameters.
- **Method Signature:** Consists of the method name and the type, order, and number of its parameters. This uniquely identifies a method.
- **Return Type:** Specifies the data type of the value that a method returns. If a method does not return a value, its return type is void.
- **void Method:** A method that performs actions but does not return any value. It does not require a return statement.
- **Method Call:** The act of invoking a method to execute its code. If a method returns a value, the call is typically part of an assignment or an expression; if it's void, it's a standalone statement.
- **Pass by Value:** Java's mechanism for passing parameters, where a copy of the actual parameter's value is made and passed to the formal parameter. Any changes made to the formal parameter inside the method do not affect the original actual parameter outside the method.

- **Runtime Stack (Call Stack / Activation Records):** A data structure used by the Java Virtual Machine (JVM) to manage method calls. When a method is called, its "activation record" (containing local variables and parameters) is "pushed" onto the stack. When the method finishes, its record is "popped" off. This helps track the flow of execution and variable visibility. The main method is always at the bottom of the stack.
- **Modularization:** A software design concept involving breaking down a program into smaller, independent, and reusable units (modules), which in Java are implemented as methods.
- **Method Overloading:** Defining multiple methods in the same class with the same name but different method signatures (i.e., different numbers, types, or order of parameters). Java determines which version to execute based on the actual parameters passed during the call.
- **Ambiguous Invocation/Call:** Occurs when the Java compiler cannot determine which overloaded method to call because the actual parameters provided could match more than one method signature, leading to a compilation error.
- **Scope of Variables (Lifetime of a Variable):** The part of the program where a variable can be referenced. A local variable's scope extends from its declaration to the end of the block or method in which it is declared. Variables declared in outer blocks are visible in nested blocks, but you cannot declare a variable with the same name in a nested block if it causes an overlap in scope.
- **Reusability:** The ability to use a method or code segment multiple times within the same program or in different programs, avoiding redundant coding.

---

**Suggestions for Mid-Term Preparation**

1. **Understand Concepts Deeply:** The exam heavily emphasizes understanding the underlying concepts. Don't just memorize syntax; understand *why* certain constructs are used and *how* they work.
2. **Focus on Key Topics:** Prioritize your study time on:
    - **Data Types:** Including primitive types (byte, short, int, long, float, double, boolean, char) and how they store values.
    - **Type Conversion (Casting):** Understand implicit (widening) and explicit (narrowing) conversions, and when data loss can occur.

- **Boolean Expressions:** How to form them using relational and logical operators, and how they evaluate to true or false.
- **Expression Evaluation & Operator Precedence:** Know the order of operations for mathematical, relational, and logical operators.
- **if and switch Statements:** Be proficient in writing and tracing simple if, if-else, nested if, and switch statements. Understand the break statement's role in switch and the potential pitfalls of semicolons after if conditions.
- **Loops (while, do-while, for):** Be able to write and trace simple for and while loops. Understand their control mechanisms, how to use sentinel values, and the implications of infinite loops, break, and continue statements.
- **Methods:** Understand how to define, call, and pass parameters to methods (both void and value-returning). Be familiar with formal vs. actual parameters, method signatures, method overloading, and the scope of variables. Also, recall that methods should be declared outside the main method.
- **Math Class Functions:** Know how to use common Math functions (e.g., Math.pow, Math.sqrt, Math.random, Math.max, Math.min).
- **Character and String Manipulation:** Understand the char type, escape sequences, and fundamental string operations (concatenation, length(), charAt(), substring(), equals(), etc.). Also, know how to use printf for formatted output.