Module I: Introduction to object-oriented programming Techniques

Course: Object oriented Programming in JAVA

Dr. Vishwa Pratap Singh

NextGen Academy

February 10, 2024

NextGen Academy
Towards fulfilling a million dreams

NextGen Academy
Towards fulfilling a million dreams

Introduction to Java

- Java is a class-based, object-oriented programming language designed for platform independence.
- Write Once, Run Anywhere (WORA): Compiled Java code can run on any platform without recompilation.
- Developed by James Gosling at Sun Microsystems in 1995, later acquired by Oracle Corporation.
- Known for simplicity, robustness, and security, making it popular for enterprise-level applications.
- Widely used for desktop, web, and mobile application development.

NextGen Academy
Towards fulfilling a million dreams

Java History

- Initiated by the Green team at Sun Microsystems in 1991, led by James Gosling.
- First public release in 1996 as Java 1.0, with subsequent versions introducing new configurations.
- Sun Microsystems released Java Virtual Machine (JVM) as open-source in 2006.
- Principles: Simple, robust, secured, high-performance, portable, and multi-threaded.
- Java's evolution involved ISO standards, free JVM release, and adoption in various domains.

Implementation of Java Application

1. **Creating the Program:** Develop Java programs using a Text Editor (Notepad) or an Integrated Development Environment (IDE) like NetBeans.
2. **Compiling the Program:** Utilize the Java Development Kit (JDK) and the Java compiler (javac) to convert source code into bytecode.
3. **Running the Program:** Execute the bytecode using the Java Interpreter, ensuring proper JDK installation and path setup.

NextGen Academy
Towards fulfilling a million dreams

Creating a Java Program

```
class Test {
    public static void main(String[] args) {
        System.out.println("My First Java Program.");
    }
};
```

File Saving: Save as `Test.java`

NextGen Academy
Towards fulfilling a million dreams

Compiling and Running a Java Program

- **Compile:** `javac Test.java`
- **Run:** `java Test`
- If successful, creates `Test.class` containing bytecode.

NextGen Academy
Towards fulfilling a million dreams

Why is it named Java?

- After the name OAK, team suggested names like Silk, Jolt, revolutionary, DNA, and dynamic.
- James Gosling chose Java while having coffee near his office.
- Java is the name of an island in Indonesia known for producing the first coffee (Java coffee).
- The name reflects the essence of technology and uniqueness.

NextGen Academy
Towards fulfilling a million dreams

Java Terminology

- **Java Virtual Machine (JVM):** Executes bytecode generated by the compiler; platform-independent.
- **Bytecode in the Development Process:** Compiled Java source code, saved as .class files.
- **Java Development Kit (JDK):** Complete development kit with compiler, JRE, debuggers, and docs.
- **Java Runtime Environment (JRE):** Allows running Java programs; includes browser, JVM, applet support, and plugins.
- **Garbage Collector:** Manages memory by recollecting unreferenced objects, enhancing Java's robustness.
- **ClassPath:** File path where the Java runtime and compiler look for .class files; includes default and external libraries.

NextGen Academy
Towards fulfilling a million dreams

Primary Features of Java

- **Platform Independent:** Bytecode runs on any platform; achieved through JVM.
- **Object-Oriented Programming (OOP):** Organizes programs as collections of objects with concepts like abstraction, encapsulation, inheritance, and polymorphism.
- **Simple:** Lacks complex features like pointers and operator overloading; no explicit memory allocation.
- **Robust:** Reliable language with features like garbage collection, exception handling, and memory allocation checks.
- **Secure:** Absence of pointers prevents out-of-bound array access; Java runs in a secure environment.

More Features of Java

- **Distributed:** Supports creating distributed applications using technologies like Remote Method Invocation (RMI) and Enterprise Java Beans (EJB).
- **Multithreading:** Allows concurrent execution for optimal CPU utilization.
- **Portable:** Code written on one machine can run on any other machine; platform-independent bytecode.
- **High Performance:** Optimized architecture, Just In Time (JIT) compiler, and reduced runtime overhead.
- **Dynamic Flexibility:** Completely object-oriented, supports dynamic class modification, and native methods.
- **Sandbox Execution:** Java programs run in a separate space, enhancing security through bytecode verification.
- **Write Once Run Anywhere:** Java applications generate platform-independent bytecode, ensuring global portability.
- **Power of Compilation and Interpretation:** Utilizes both compilation and interpretation for flexibility and performance.

Java Language Specification, API, JDK, and IDE

- **Java Language Specification:** Defines syntax and semantics; available at `docs.oracle.com/javase/specs/`.
- **Java API:** Contains predefined classes and interfaces; expanding with each release; available at `download.java.net/jdk8/docs/api/`.
- **Java Development Kit (JDK):** Software for compiling and running Java programs; different editions for various applications (Java SE, Java EE, Java ME).
- **Integrated Development Environment (IDE):** Tools like NetBeans, Eclipse, and TextPad provide a graphical interface for development tasks.

NextGen Academy
Towards fulfilling a million dreams

Overview

- Java's two-step execution process
- Compilation: Translates '.java' to bytecode
- Execution: Runs bytecode on the JVM

Compilation: Steps

- Parse: Convert '*.java' to AST-Nodes
- Enter: Add definitions to symbol table
- Process annotations: Handle user-defined annotations
- Attribute: Perform name resolution, type checking, constant folding
- Flow: Analyze dataflow, check assignments and reachability
- Desugar: Remove syntactic sugar
- Generate: Produce '.class' files

NextGen Academy
Towards fulfilling a million dreams

Execution: Class Loader

- Load main class and referenced classes into memory
- Class Loader types: primordial (default), non-primordial (user-defined)

NextGen Academy
Towards fulfilling a million dreams

Execution: Bytecode Verifier

- Inspect loaded bytecode for security
- Verify variables initialization, method call types, access rules
- Prevent stack overflow and other runtime issues

NextGen Academy
Towards fulfilling a million dreams

Execution: Just-In-Time Compiler

- Convert bytecode to machine code at runtime
- Enhance execution speed by avoiding repeated interpretation
- Performance gains for frequently executed methods

NextGen Academy
Towards fulfilling a million dreams

Conclusion

- Java's platform independence due to two-step execution
- Trade-off: Execution time vs. platform independence
- Example: Simple printing program

NextGen Academy
Towards fulfilling a million dreams

Implementation

- Steps to create, compile, and execute a Java program
- Illustration with a sample code and terminal commands

Contents

NextGen Academy
Towards fulfilling a million dreams

Introduction to IDEs

- IDE importance: Simplifying development
- Tools provided: Compiler, interpreter, profiler, debugger, syntax checker, auto-correction

Setting Up NetBeans IDE

- Download NetBeans from official website
- Installation steps based on platform
- Starting NetBeans and creating a new Java project

NextGen Academy
Towards fulfilling a million dreams

Contents

NextGen Academy
Towards fulfilling a million dreams

Identifiers

- Definition: Names for elements like classes, methods, variables
- Rules: Alphanumeric characters, underscores, and dollar signs; must start with a letter; case-sensitive

NextGen Academy
Towards fulfilling a million dreams

Variables

- Definition: Represent values; containers for data
- Declaration: Specify type and name (e.g., int count)
- Initialization: Assign initial values
- Types: int, double, boolean, etc.

Assignment Statements

- Role: Designate a value for a variable
- Syntax: variable = expression
- Examples: int x = 1; double radius = 1.0; x = y + 1;
- Expressions: Computations involving values, variables, operators

NextGen Academy
Towards fulfilling a million dreams

Numeric Literals

- Literal: Constant value assigned to a variable.
- Represents boolean, numeric, character, or string data.
- Example: `int x = 100;`
- By default, every literal is of int type.
- Provides a synthetic representation of data.
- Used for expressing values in programs.
- Importance in initializing variables with specific values.
- Follows syntax conventions for different data types.
- Enhances code readability and maintenance.
- Example program illustrates various numeric literals.

Integral Literals

- Four ways to specify for integral data types.
- Decimal literals (Base 10): `int x = 101;`
- Octal literals (Base 8): `int x = 0146;`
- Hexa-decimal literals (Base 16): `int x = 0X123Face;`
- Binary literals: `int x = 0b1111;`
- Example program illustrating integral literals.
- Literals can be explicitly specified for byte and short indirectly.
- Compiler treats integral literals as int by default.
- Byte and short literals can be specified within their ranges.
- Understanding the significance of prefixes (0, 0X, 0b).

NextGen Academy
Towards fulfilling a million dreams

Floating-Point Literal

- Floating-point literals only in decimal form.
- Decimal literals (Base 10): `double d = 123.456;`
- Example program for floating-point literals.
- Default: Every floating-point literal is of double type.
- Suffix `f` or `F` for float type.
- Explicitly specifying floating-point literal type.
- Importance of suffixes `f`, `F`, `d`, `D`.
- Common error: malformed floating-point literals.
- Enhances precision and control over numeric values.

NextGen Academy
Towards fulfilling a million dreams

Evaluating Expressions and Operator Precedence

- Expression evaluation with parentheses.
- Infix, Prefix, and Postfix notations.
- Shunting Yard Algorithm by Edgar Dijkstra.
- Algorithm steps for expression evaluation.
- Importance of converting infix to postfix notation.
- Utilizing two stacks for operands and operators.
- Handling parentheses and operators in the algorithm.
- Example of converting and evaluating an expression.
- Significance in compiler design and mathematical computations.

Operator Precedence (Contd.)

- Operator precedence and associativity.
- Java Operator Precedence Table.
- Examples of unary and binary operators.
- Precedence example: `1 + 5 * 3`.
- Understanding how higher precedence affects evaluation.
- Associativity importance in expressions with the same precedence.
- Clearing misconceptions about operator behavior.
- Real-world examples of operator precedence in programming.
- Building a strong foundation for expression evaluation.

NextGen Academy
Towards fulfilling a million dreams

Increment and Decrement Operators

- Unary operators for increasing or decreasing a variable.
- Unary operators can be applied to all primitive types except Boolean.
- Pre Increment Operator (++x).
- Post Increment Operator (x++).
- Examples and output explanations.
- Practical scenarios for using increment and decrement.
- Implications of using these operators in loops and conditions.
- Avoiding common pitfalls and ensuring code clarity.
- Building efficient and concise code with increment operators.

NextGen Academy
Towards fulfilling a million dreams

Pre Increment Operator

- Syntax: `++x`.
- Increases the value before assignment.
- Example program and output.
- Common use cases for pre increment.
- Influence on variable value and subsequent operations.
- When to choose pre increment for optimal code.
- Demonstrating the flow of execution with pre increment.
- Advantages in specific programming scenarios.

Post Increment Operator

- Syntax: `x++`.
- Increases the value after assignment.
- Example program and output.
- Common use cases for post increment.
- Influence on variable value and subsequent operations.
- When to choose post increment for optimal code.
- Demonstrating the flow of execution with post increment.
- Advantages in specific programming scenarios.

NextGen Academy
Towards fulfilling a million dreams

Example Program for Increment Operators

- Demonstrates the use of both pre and post increment operators.
- Shows the difference in behavior.
- Analyzing the output and variable values at each step.
- Emphasizing the importance of choosing the right operator.
- Practical implications in real-world coding scenarios.
- Encouraging efficient and readable code practices.
- Enhancing problem-solving skills with increment operators.

Pre Decrement Operator

- Decrement operator decreases the value by 1.
- Types: Pre decrement (–x).
- Example: `int y = --x;`
- Syntax: `--x`
- Output: `y value is:  9`

NextGen Academy
Towards fulfilling a million dreams

Post Decrement Operator

- Types: Post decrement (x–).
- Example: `int y = x--;`
- Syntax: `x--`
- Output: `y value is: 10`

NextGen Academy
Towards fulfilling a million dreams

Limitations of Inc/Dec Operators

- Applicable only to variables.
- Compile-time error for constants.
- No nesting allowed.
- Cannot be used on final variables.
- Not applicable to boolean types.

Numeric Type Conversions Overview

- Conversion of one data type to another.
- Automatic conversion for compatible types.
- Widening: `byte -> short -> int -> long -> float -> double`.
- Explicit casting for narrowing.

NextGen Academy
Towards fulfilling a million dreams

Converting Between Numeric Types

- Automatic type conversion example.
- Widening pattern.
- Explicit type casting example.

Converting Strings to Numeric Types

- String to numeric conversion.
- Number subclasses: Byte, Integer, Double, etc.
- Example using `Integer.parseInt()`.
- Example using `Float.parseFloat()`.

NextGen Academy
Towards fulfilling a million dreams

Converting Numeric Types Into Strings

- Converting numbers to strings.
- Automatic conversion through concatenation.
- `String.valueOf()` method.
- `.toString()` method for Number subclasses.

NextGen Academy
Towards fulfilling a million dreams