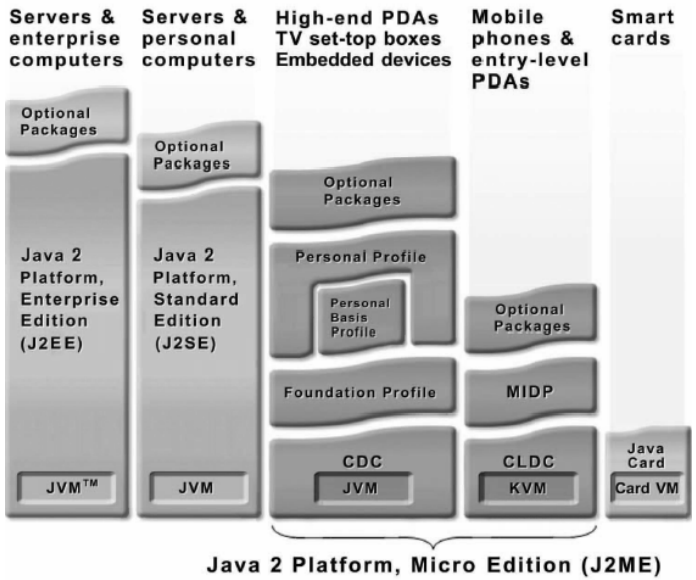
Introduction to Visual Basic and Visual C++

Lesson 13

Introduction to Java

Overview

Overview



JDK Editions

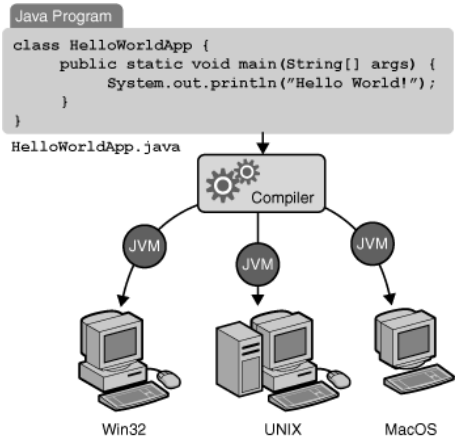
- Before you can write and run the simple Java program in this lesson, you need to install the Java platform on your computer system.
  - Java Standard Edition (J2SE)
    - J2SE can be used to develop client-side standalone applications or applets.
  - Java Enterprise Edition (J2EE)
    - J2EE can be used to develop server-side applications such as Java servlets and Java ServerPages.
  - Java Micro Edition (J2ME)
    - J2ME can be used to develop applications for mobile devices such as cell phones.

Overview of Software Development Process

- In the Java programming language, all source code is first written in plain text files ending with the **.java** extension.
- Those source files are then compiled into **.class** files by the **javac** compiler.
- A **.class** file does not contain code that is native to your processor; it instead contains bytecodes — the machine language of the Java Virtual Machine1 (JVM).
- The java launcher tool then runs your application with an instance of the Java Virtual Machine.

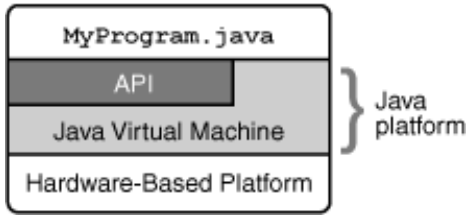
Java Virtual Machine (JVM)

- Through the JVM, the same application is capable of running on multiple platforms



Java Platform

- The Java platform has two components:
  - The Java Virtual Machine (JVM)
  - The Java Application Programming Interface (API)
- The API and Java Virtual Machine insulate the program from the underlying hardware



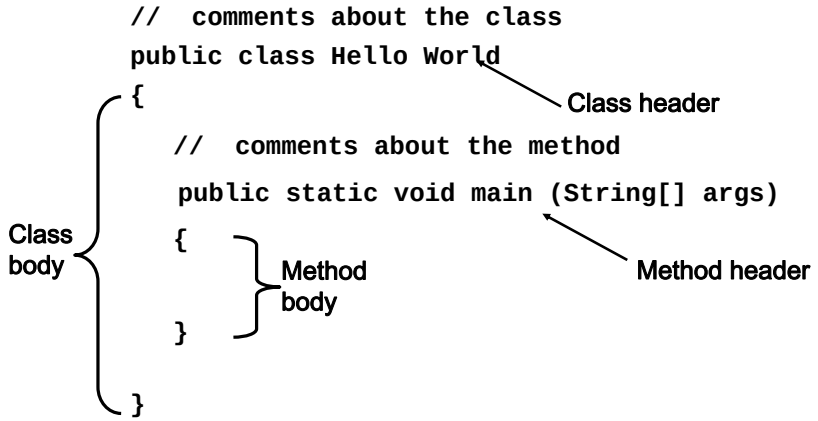
My First Program

Hello World

Java Program Structure

- In the Java programming language:
  - A program is made up of one or more classes
  - A class contains one or more methods
  - A method contains program statements
  - Program statements can reference local or instance variables. There is no concept of global variable.
- These terms will be explored in detail throughout the course
- A Java application always contains a method called main

Java Program Structure



Comments

- Comments in a program are called inline documentation
- They should be included to explain the purpose of the program and describe processing steps
- They do not affect how a program works
- Java comments can take three forms:

```
// this comment runs to the end of the line

/*  this comment runs to the terminating
    symbol, even across line breaks       */

/** this is a javadoc comment   */
```


Identifiers

- *Identifiers* are the words a programmer uses in a program
- An 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
- By convention, programmers use different case styles for different types of identifiers, such as
  - *title case* for class names - `Lincoln`
  - *upper case* for constants - `MAXIMUM`

Java Programming Foundation

A Quick Look to Java

Grouping Classes: The Java API

- API = *Application Programming Interface*
- Java = small core + extensive collection of packages
- A ***package*** consists of some related Java classes:
  - Swing: a GUI (graphical user interface) package
  - AWT: Application Window Toolkit (more GUI)
  - util: utility data structures
- The ***import*** statement tells the compiler to make available classes and methods of another package
- A ***main*** method indicates where to begin executing a class (if it is designed to be run as a program)

Comments

- In Java, comments are preceded by:
  - Two slashes (//) in a line, or
  - Enclosed between /* and */ in one or multiple lines.
- When the compiler sees //, it ignores all text after // in the same line.
- When it sees /*, it scans for the next */ and ignores any text between /* and */.

Reserved Words

- Reserved words or keywords are words that have a specific meaning to the compiler and cannot be used for other purposes in the program.
- For example, when the compiler sees the word **class**, it understands that the word after class is the name for the class.

Here's a list of keywords in the Java programming language. You cannot use any of the following as identifiers in your programs. The keywords `const` and `goto` are reserved, even though they are not currently used. `true`, `false`, and `null` might seem like keywords, but they are actually literals; you cannot use them as identifiers in your programs.

`abstract`	`continue`	`for`	`new`	`switch`
`assert`***	`default`	`goto`*	`package`	`synchronized`
`boolean`	`do`	`if`	`private`	`this`
`break`	`double`	`implements`	`protected`	`throw`
`byte`	`else`	`import`	`public`	`throws`
`case`	`enum`****	`instanceof`	`return`	`transient`
`catch`	`extends`	`int`	`short`	`try`
`char`	`final`	`interface`	`static`	`void`
`class`	`finally`	`long`	`strictfp`**	`volatile`
`const`*	`float`	`native`	`super`	`while`

* not used
** added in 1.2
*** added in 1.4
**** added in 5.0

Modifiers

- Java uses certain reserved words called modifiers that specify the properties of the data, methods, and classes and how they can be used. Examples of modifiers are public and static. Other modifiers are private, final, abstract, and protected.
  - A public datum, method, or class can be accessed by other programs.
  - A private datum or method cannot be accessed by other programs.

References and Primitive Data Types

- Java distinguishes two kinds of entities
  - Primitive types
  - Objects
- Primitive-type data is stored in primitive-type variables
- Reference variables store the *address of* an object
  - No notion of "object (physically) in the stack"
  - No notion of "object (physically) within an object"

Primitive Data Types

- Represent numbers, characters, boolean values
- Integers: byte, short, int, and long
- Real numbers: float and double
- Characters: char

Data Type	Range of Values
`byte`	-128 .. 127 (8 bits)
`short`	-32,768 .. 32,767 (16 bits)
`int`	-2,147,483,648 .. 2,147,483,647 (32 bits)
`long`	-9,223,372,036,854,775,808 .. ... (64 bits)
`float`	$+/-10^{-38}$ to $+/-10^{+38}$ and 0, about 6 digits precision
`double`	$+/-10^{-308}$ to $+/-10^{+308}$ and 0, about 15 digits precision
`char`	Unicode characters (generally 16 bits per char)
`boolean`	True or false

Equality and Relational Operators

Standard algebraic equality or relational operator	Java equality or relational operator	Sample Java condition	Meaning of Java condition
Equality operators			
=	==	x == y	x is equal to y
≠	!=	x != y	x is not equal to y
Relational operators			
>	>	x > y	x is greater than y
<	<	x < y	x is less than y
≥	>=	x >= y	x is greater than or equal to y
≤	<=	x <= y	x is less than or equal to y

Operators

- subscript [ ], call ( ), member access .
- pre/post-increment ++ --, boolean complement !, bitwise complement ~, unary + -, type cast (type), object creation new
- * / %
- binary + - (+ also concatenates strings)
- signed shift << >>, unsigned shift >>>
- comparison < <= > >=, class test instanceof
- equality comparison == !=
- bitwise and &
- bitwise or |
- logical (sequential) and &&
- logical (sequential) or ||
- conditional cond ? true-expr : false-expr
- assignment =, compound assignment += -= *= /= <<= >>= >>>= &= |=

Introduction to Object

All Java programs are built from classes

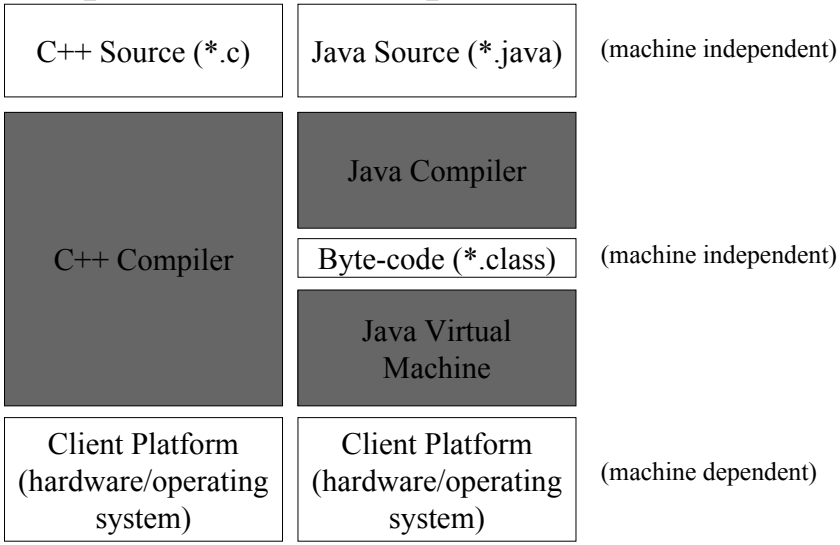
Classes

- The class is the essential Java construct. A class is a template or blueprint for objects.
- To program in Java, you must understand classes and be able to write and use them.
- For now, though, understand that a program is defined by using one or more classes.

What is Object?

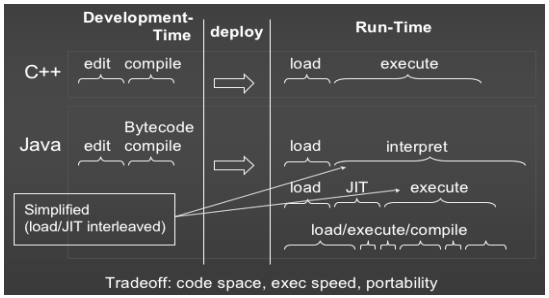
- Object is a collection of data along with the functions to work with that data as opposed to purely procedural languages where the data and functions are separate, and the data is passed to the function to work with
- In Java, all code must be written inside object definitions
  - A **class** is an object definition, containing the data and function elements necessary to create an object
  - An **instance** of an object is one object created from the class definition (the process is known as instantiation)
  - The data and function elements are known as **members**
  - Data members are also known as **fields**, **properties**, or **attributes**, and function members as **methods**

Interpretation vs Compilation

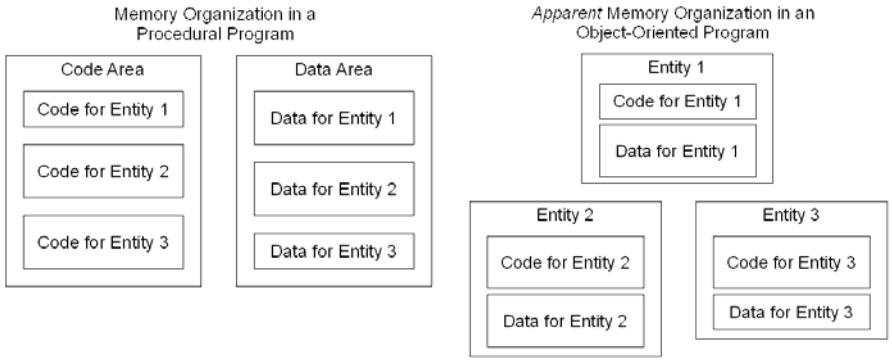


JIT Compilers

- The trade-off in execution speed can be mitigated by adding a just-in-time (JIT) compiler to the virtual machine.
- Preserves portability, achieves near native performance after first references
- Compiling "on-the-fly" makes initial run slower.

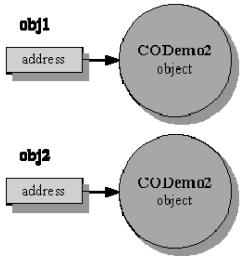


Memory Organization between Procedural Program and OO Program



Benefit of Classes

- Classes simplify programming, because the client can use only the `public` methods exposed by the class. Such methods are usually client oriented rather than implementation oriented.
  - Clients are neither aware of, nor involved in, a class's implementation.
  - Clients generally care about *what* the class does but not *how* the class does it.



Method

- Methods that modify the values of `private` variables should verify that the intended new values are proper.
- If they are not, the set methods should place the `private` variables into an appropriate consistent state.

Interface

- Interfaces change less frequently than implementations.
- When an implementation changes, implementation-dependent code must change accordingly.
- Hiding the implementation reduces the possibility that other program parts will become dependent on class-implementation details.

Instance

- Instance variables
  - Can be initialized when they are declared or in a constructor
  - Should maintain consistent (valid) values
- If a class does not define a constructor the compiler will provide a default constructor
- `public` services (or `public` interface)
  - `public` methods available for a client to use

Controlling Access to Members

- A class's public interface
  - `public` methods a view of the services the class provides to the class's clients
- A class's implementation details
  - `private` variables and `private` methods are not accessible to the class's clients
- This object-oriented concept is called "Encapsulation".

Referring to the Current Object's Members with the this Reference

- The `this` reference
  - Any object can access a reference to itself with keyword `this`
  - Non-`static` methods implicitly use `this` when referring to the object's instance variables and other methods
  - Can be used to access instance variables when they are shadowed by local variables or method parameters
- A `.java` file can contain more than one class
  - But only one class in each `.java` file can be `public`

Notes on Set and Get Methods

- *Set* methods
  - Also known as mutator methods
  - Assign values to instance variables
  - Should validate new values for instance variables
    - Can return a value to indicate invalid data
- *Get* methods
  - Also known as accessor methods or query methods
  - Obtain the values of instance variables
  - Can control the format of the data it returns

Notes on Set and Get Methods

- Predicate methods
  - Test whether a certain condition on the object is true or false and returns the result
  - Example: an `isEmpty` method for a container class (a class capable of holding many objects)
  - Predicate methods always return a boolean data type.
  - Clever naming of predicate methods can make your code easier to understand
    `if (myStack.isEmpty())`
- Encapsulating specific tasks into their own methods simplifies debugging efforts

Composition

- Composition
  - A class can have references to objects of other classes as members
  - Sometimes referred to as a *has-a* relationship
- One form of software reuse is composition, in which a class has as members references to objects of other classes.

Final Instance Variables

- Principle of least privilege
  - Code should have only the privilege and access it needs to accomplish its task, but no more
- `final` instance variables
  - Keyword `final`
    - Specifies that a variable is not modifiable (is a constant)
  - `final` instance variables can be initialized at their declaration
    - If they are not initialized in their declarations, they must be initialized in all constructors

Static Class Members

- `static` fields
  - Also known as class variables
  - Represents class-wide information
  - Used when:
    - All objects of the class should share the same copy of this instance variable or
    - This instance variable should be accessible even when no objects of the class exist
  - Can be accessed with the class name or an object name and a dot (`.`)
  - Must be initialized in their declarations, or else the compiler will initialize it with a default value (`0` for `int`s)

Garbage Collection and Method finalize

- Garbage collection
  - JVM marks an object for garbage collection when there are no more references to that object
  - JVM's garbage collector will retrieve those objects memory so it can be used for other objects
- `finalize` method
  - All classes in Java have the `finalize` method
    - Inherited from the `Object` class
  - `finalize` is called by the garbage collector when it performs termination housekeeping
  - `finalize` takes no parameters and has return type `void`

Static Class Members

- `String` objects are immutable
  - `String` concatenation operations actually result in the creation of a new `String` object
- `static` method `gc` of class `System`
  - Indicates that the garbage collector should make a best-effort attempt to reclaim objects eligible for garbage collection
  - It is possible that no objects or only a subset of eligible objects will be collected
- `static` methods cannot access non-`static` class members
  - Also cannot use the `this` reference

Static Import

- `static` import declarations
  - Enables programmers to refer to imported `static` members as if they were declared in the class that uses them
  - Single `static` import
    - `import static packageName.ClassName.staticMemberName;`
  - `static` import on demand
    - `import static packageName.ClassName.*;`
    - Imports all `static` members of the specified class

Enumerations

- `enum` types
  - Declared with an `enum` declaration
    - A comma-separated list of `enum` constants
    - Declares an `enum` class with the following restrictions:
      - `enum` types are implicitly `final`
      - `enum` constants are implicitly `static`
      - Attempting to create an object of an `enum` type with `new` is a compilation error
  - `enum` constants can be used anywhere constants can
  - `enum` constructor
    - Like class constructors, can specify parameters and be overloaded

Enumerations (Cont.)

- `static` method `values`
  - Generated by the compiler for every `enum`
  - Returns an array of the `enum`'s constants in the order in which they were declared
- `static` method `range` of class `EnumSet`
  - Takes two parameters, the first and last `enum` constants in the desired range
  - Returns an `EnumSet` containing the constants in that range, inclusive
  - An enhanced `for` statement can iterate over an `EnumSet` as it can over an array