CMSC 131

Object-Oriented Programming I

Java Introduction

Dept of Computer Science
University of Maryland College Park

This material is based on material provided by Ben Bederson, Bonnie Dorr, Fawzi Emad, David Mount, Jan Plane

Overview

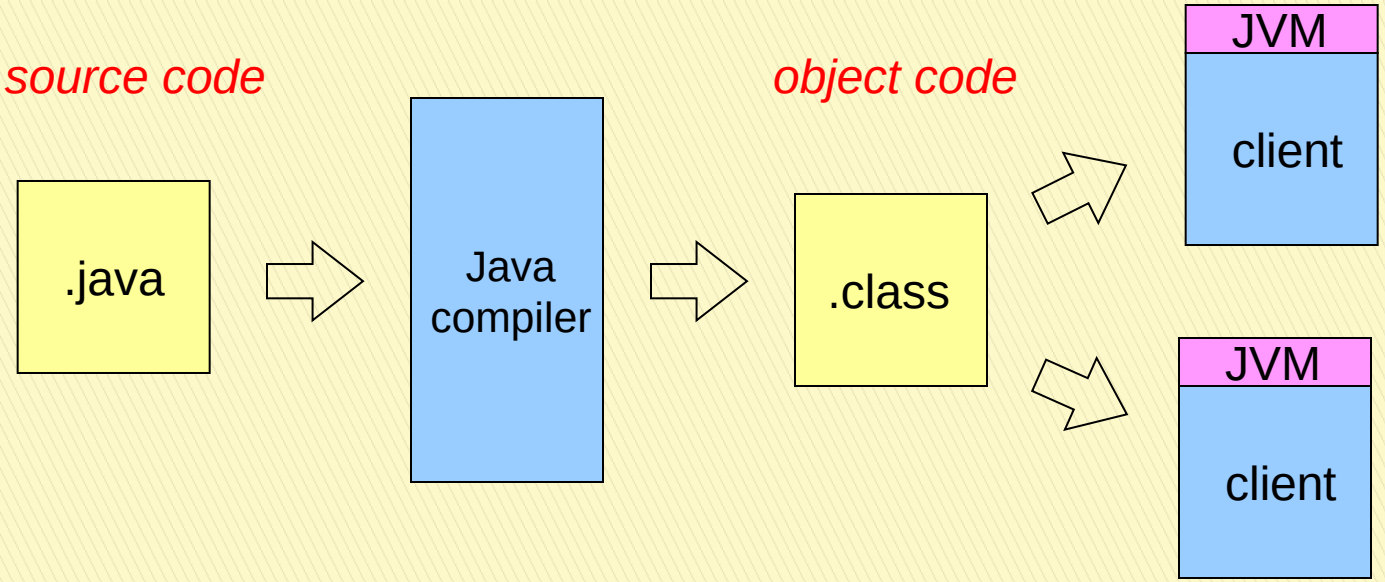
- Java Programming Language
- Basic Java Concepts
- Sample Java Program

Java Programming Language

- **Java** – Why is it special?
  - An **object-oriented** programming language. (More about this later.)
  - Developed in early 1990's by James Gosling at Sun Microsystems for controlling programmable devices.
  - Unlike C++, Java programs are **portable** between machines – why?
    - C/C++ programs are compiled into **machine code** and is executed directly by CPU. Different machines may behave differently.
    - Java programs are compiled into **Java bytecode**. This is a machine code for a "virtual computer" called the **Java Virtual Machine** (JVM).
    - A JVM **interpreter** reads bytecode and simulates its execution.
  - Has a rich set of **libraries** that allow you to create graphics, communicate over networks, interact with databases, etc.
  - The Java compiler and interpreter are part of **Java Software Development Kit** (SDK), also referred to as **Java Development Kit** (JDK).

Java Virtual Machine

- Java includes definition of *Java bytecode* = "fake" machine code for Java
- Java compilers produce Java bytecode
- To run Java bytecode, must have bytecode interpreter ("Java Virtual Machine") on client machine



- For efficiency, JVMs often compile bytecode into native machine code
- There are also "native" Java compilers (these compile Java directly to machine code)

Java Programming Concepts

- A quick overview of fundamental Java concepts. (More details later.)

  Object →The principal entities that are manipulated by a Java program

  Class → A **blueprint** or template for the structure of an object

  Method → Java's term for a **procedure** or **subroutine**. Every method belongs to some class, and a class may have many methods

  Main Method: There must be exactly one method called **main**. This is where execution begins

  Statements: Individual instructions. Examples:

  Declarations → declare (and initialize) variables

  Assignment → compute and assign a new value to a variable

  Method invocation → execute a method (which may return a result)

  Control flow → alter the order in which statements are executed

Sample Program

```
/**
 * My first sample Java program.
 */
public class MyFirstProgram {

        public static void main( String[ ] args ) {
                int secondsPerMinute = 60;                          // let's create some variables
                int minutesPerLecture = 50;
                int totalSeconds = secondsPerMinute * minutesPerLecture;
                System.out.println( "There are " + totalSeconds + " seconds in a lecture.“ );
        }

}
```

Program Output

There are 3000 seconds in a lecture.

- Let's look at each component of this program

Java Program Organization

- **Class:** The structure around which all Java programs are based. A typical Java program consists of many classes, and typically, each class resides in its own file, whose name is based on the class's name. The class is delimited by curly braces { … }.

File name: **MyFirstProgram.java**:

```
public class MyFirstProgram {
        … (contents of the class go here) …
}
```

- A class consist of a combination of **data units**, called **variables**, and **computation units**, called **methods**.

Java Program Organization

- **Methods:** This is where all the computation takes place. Each method has a name, a list of arguments enclosed in parentheses, and body, enclosed in curly braces.

```
public static void main( String[ ] args ) {
        ... (contents of the main method go here) ...
}
```

- **Variables**: These are the data items that the methods operate on. Variables can be of various types, integers, for example:

```
int secondsPerMinute = 60;
int minutesPerLecture = 50;
```

Java Program Organization

- **Statements**: Each method consists of some number of statements. There are many different types of statements, which perform diverse tasks:

  Declarations → specify variable types (and optionally initialize)

```
int x, y, z;                // three integer variables
String s = "Howdy";         // a character string variable
boolean isValid = true;     // a boolean (true or false) variable
```

  Assignments → assign variables new values

```
x = 13;
```

  Method invocation → call other methods

```
System.out.println( "Print this message" );
```

  Control flow → determine the order of statement execution. (These include **if-then-else**, **while**, **do-while**, **for**. We'll cover them later.)

- **Expressions/Operators** → Java provides various operators, which allow you to manipulate the variables.

```
x = (2*y – z)/32;
```

Other Java Elements: Comments

- **Comments**: are used to indicate the programmer's intent. They do not affect the program's execution, (ignored by compilear) but are an important part of any program. Two ways:

  Line comments: After **//** the rest of the line is a comment. These are often used to explain a single line of code

  Block comments: The text between any pair: **/*** … ***/** this can run over many lines or on a single line

- Comments are essential for good programming

Debugging Java Programs

- Types of errors
  - "Compile time" → caught by Eclipse / Java compiler
    - *Syntax* errors
    - Disobeys the rules of the language; violates language's grammar
    - *Type* errors: misuse of variables
  - "Run time" → appear during program execution
    - Semantic errors
    - obeys the rules of the language but does not express them meaning you intended;
    - Division by 0
    - Crash or hang or wrong outputs (because of mistakes in programming)
- Eclipse helps catch compile time errors
  - Red → error
  - Yellow → warning
- Debugging
  - Process of finding and fixing problems
  - To minimize debugging frustration → use "unit" testing
    - Write a small part, thoroughly test it, cycle back