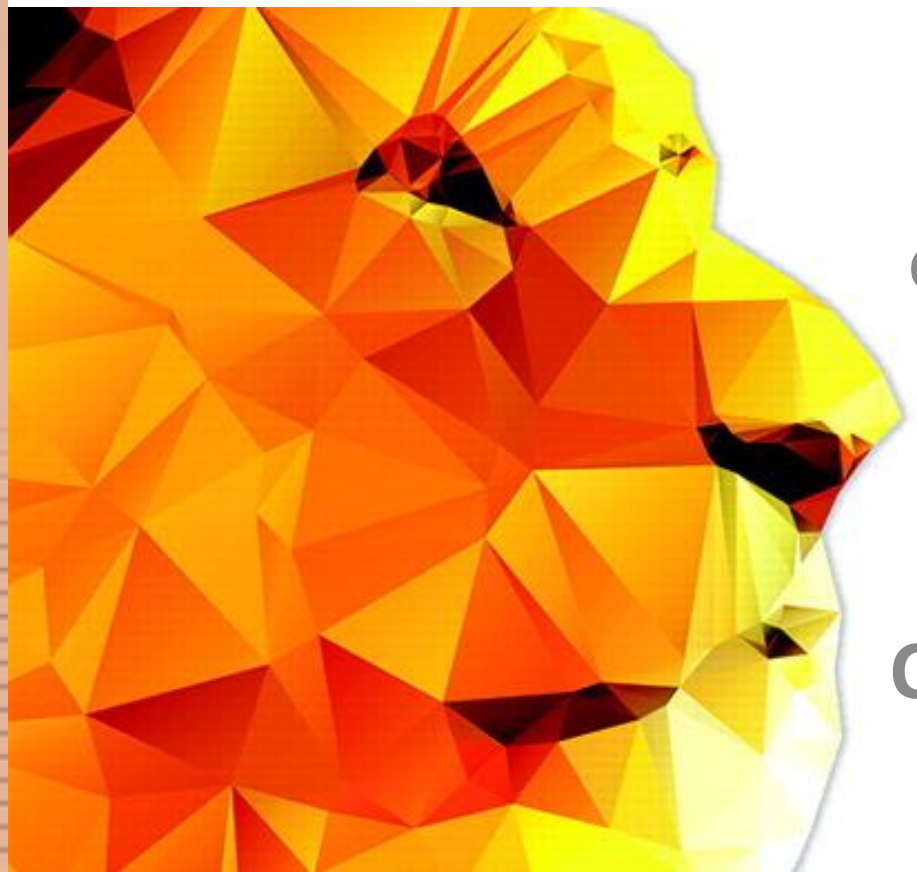
Lecture 1

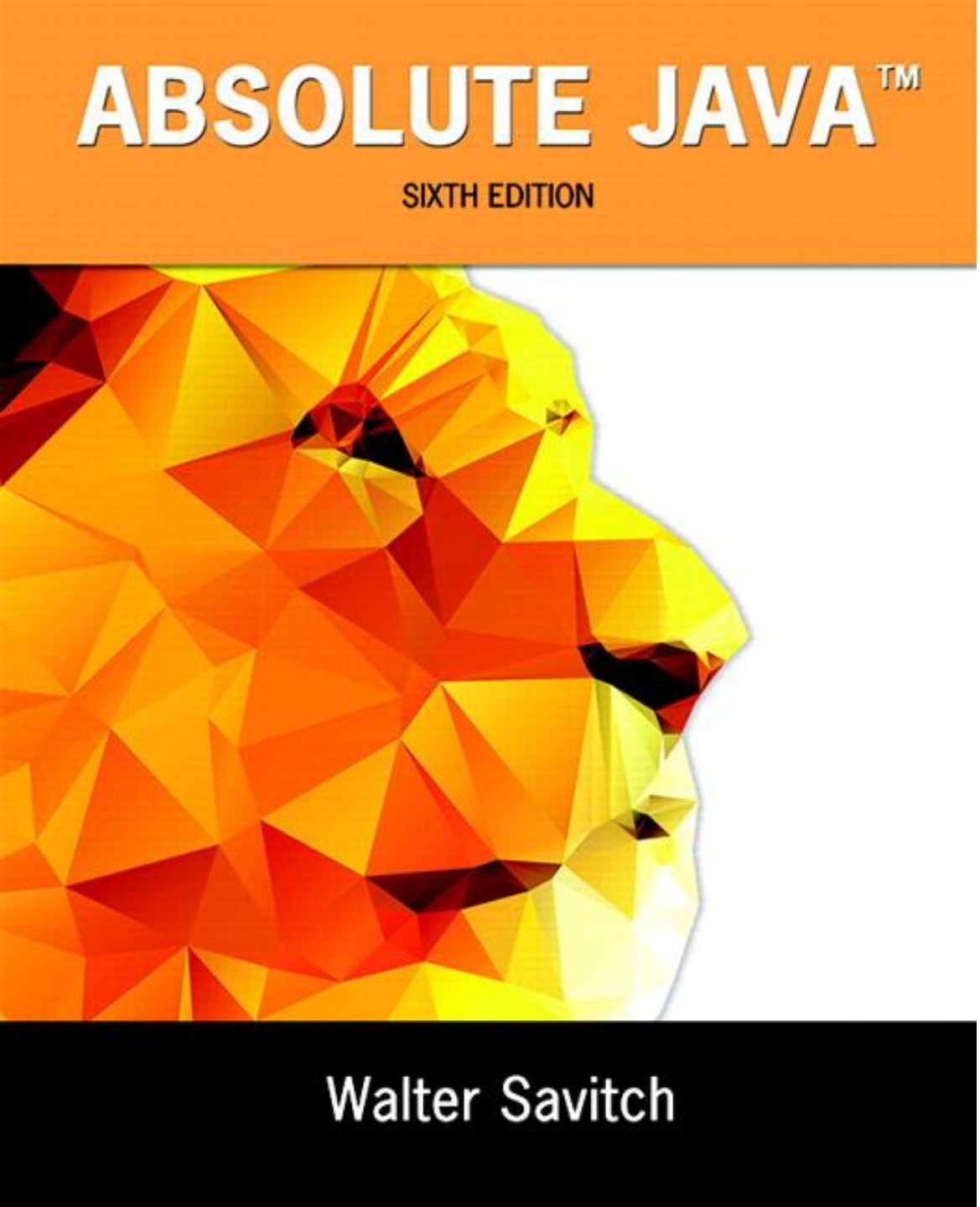


University of Hail

College of Computer Science and Engineering

CSCE 101

Computer Programming I



Chapter 1

Getting Started

Slides prepared by Rose Williams,
Binghamton University

Kenrick Mock, *University of Alaska Anchorage*

PEARSON

Introduction To Java

- Most people are familiar with Java as a language for Internet applications
- We will study Java as a general purpose programming language
  - The syntax of expressions and assignments will be similar to that of other high-level languages
  - Details concerning the handling of strings and console output will probably be new

Origins of the Java Language

- Created by Sun Microsystems team led by James Gosling in 1991 (now owned by Oracle)
- Originally designed for programming home appliances
  - Difficult task because appliances are controlled by a wide variety of computer processors
  - Team developed a two-step translation process to simplify the task of compiler writing for each class of appliances

Origins of the Java Language

- Significance of Java translation process
  - Writing a compiler (translation program) for each type of appliance processor would have been very costly
  - Instead, developed intermediate language that is the same for all types of processors : Java *byte-code*
  - Therefore, only a small, easy to write program was needed to translate byte-code into the machine code for each processor

Origins of the Java Language

- Patrick Naughton and Jonathan Payne at Sun Microsystems developed a Web browser that could run programs over the Internet (1994)
  - Beginning of Java's connection to the Internet
  - Original browser evolves into *HotJava*
- Netscape made its Web browser capable of running Java programs (1995)
  - Other companies follow suit

Objects and Methods

- Java is an *object-oriented programming (OOP)* language
  - Programming methodology that views a program as consisting of *objects* that interact with one another by means of actions (called *methods*)
  - Objects of the same kind are said to have the same *type* or be in the same *class*

Java Application Programs

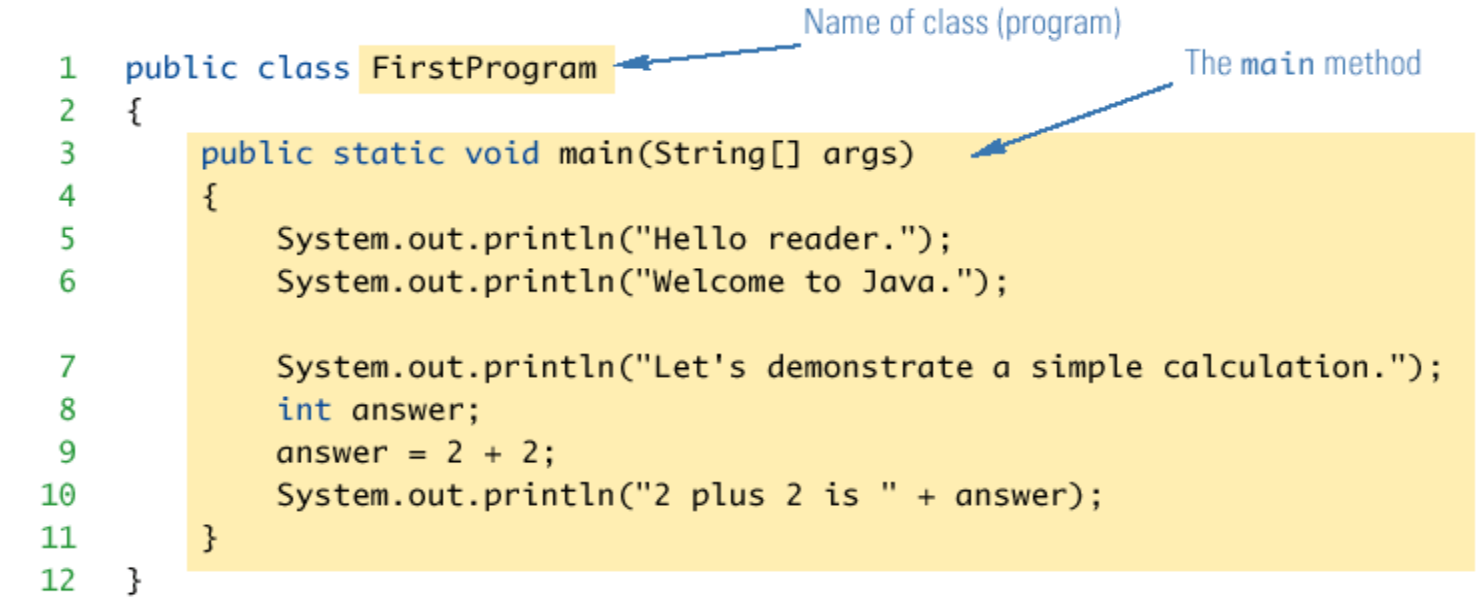
- Two common types of Java programs are *applications* and *applets*
- A Java *application program* or "regular" Java program is a class with a method named **`main`**
  - When a Java application program is run, the *run-time system* automatically invokes the method named **`main`**
  - All Java application programs start with the **`main`** method

Applets

- A Java *applet* (*little Java application*) is a Java program that is meant to be run from a Web browser
  - Can be run from a location on the Internet
  - Can also be run with an applet viewer program for debugging
  - Applets always use a windowing interface
- In contrast, application programs may use a windowing interface or console (i.e., text) I/O

A Sample Java Application Program

Display 1.1 A Sample Java Program




```
1   public class FirstProgram
2   {
3       public static void main(String[] args)
4       {
5           System.out.println("Hello reader.");
6           System.out.println("Welcome to Java.");
7           System.out.println("Let's demonstrate a simple calculation.");
8           int answer;
9           answer = 2 + 2;
10          System.out.println("2 plus 2 is " + answer);
11      }
12  }
```

SAMPLE DIALOGUE 1

```
Hello reader.
Welcome to Java.
Let's demonstrate a simple calculation.
2 plus 2 is 4
```

`System.out.println`

- Java programs work by having things called *objects* perform actions
  - **`System.out`**: an object used for sending output to the screen
- The actions performed by an object are called *methods*
  - **`println`**: the method or action that the **`System.out`** object performs

System.out.println

- *Invoking* or *calling* a method: When an object performs an action using a method
  - Also called *sending a message* to the object
  - Method invocation syntax (in order): an object, a dot (period), the method name, and a pair of parentheses
  - Arguments: Zero or more pieces of information needed by the method that are placed inside the parentheses

```
System.out.println("This is an argument");
```

Variable declarations

- Variable declarations in Java are similar to those in other programming languages
  - Simply give the type of the variable followed by its name and a semicolon

```
int answer;
```

Using = and +

- In Java, the equal sign (=) is used as the *assignment operator*
  - The variable on the left side of the assignment operator is *assigned the value* of the expression on the right side of the assignment operator

```
answer = 2 + 2;
```

- In Java, the plus sign (+) can be used to denote addition (as above) or *concatenation*
  - Using +, two strings can be connected together

```
System.out.println("2 plus 2 is " + answer);
```

Computer Language Levels

- *High-level language*: A language that people can read, write, and understand
  - A program written in a high-level language must be translated into a language that can be understood by a computer before it can be run
- *Machine language*: A language that a computer can understand
- *Low-level language*: Machine language or any language similar to machine language
- *Compiler*: A program that translates a high-level language program into an equivalent low-level language program
  - This translation process is called *compiling*

Byte-Code and the Java Virtual Machine

- The compilers for most programming languages translate high-level programs directly into the machine language for a particular computer
  - Since different computers have different machine languages, a different compiler is needed for each one
- In contrast, the Java compiler translates Java programs into *byte-code*, a machine language for a fictitious computer called the *Java Virtual Machine*
  - Once compiled to *byte-code*, a Java program can be used on any computer, making it very portable

Byte-Code and the Java Virtual Machine

- *Interpreter:* The program that translates a program written in Java byte-code into the machine language for a particular computer when a Java program is executed
  - The interpreter translates and immediately executes each byte-code instruction, one after another
  - Translating byte-code into machine code is relatively easy compared to the initial compilation step
- Most Java programs today run using a Just-In-Time or JIT compiler which compiles a section of byte-code at a time into machine code

Program terminology

- *Code*: A program or a part of a program
- *Source code* (or *source program*): A program written in a high-level language such as Java
  - The input to the compiler program
- *Object code*: The translated low-level program
  - The output from the compiler program, e.g., Java byte-code
  - In the case of Java byte-code, the input to the Java byte-code interpreter

Class Loader

- Java programs are divided into smaller parts called *classes*
  - Each class definition is normally in a separate file and compiled separately
- *Class Loader*: A program that connects the byte-code of the classes needed to run a Java program
  - In other programming languages, the corresponding program is called a *linker*

Compiling a Java Program or Class

- Each class definition must be in a file whose name is the same as the class name followed by `.java`
  - The class `FirstProgram` must be in a file named `FirstProgram.java`
- Each class is compiled with the command `javac` followed by the name of the file in which the class resides

```
javac FirstProgram.java
```

  - The result is a byte-code program whose filename is the same as the class name followed by `.class`

```
FirstProgram.class
```

Running a Java Program

- A Java program can be given the *run command* (**`java`**) after all its classes have been compiled
  - Only run the class that contains the **`main`** method (the system will automatically load and run the other classes, if any)
  - The **`main`** method begins with the line:

```
public static void main(String[ ] args)
```

  - Follow the run command by the name of the class only (no **`.java`** or **`.class`** extension)

```
java FirstProgram
```

Syntax and Semantics

- *Syntax*: The arrangement of words and punctuations that are legal in a language, the *grammar rules* of a language
- *Semantics*: The meaning of things written while following the syntax rules of a language

Tip: Error Messages

- *Bug*: A mistake in a program
  - The process of eliminating bugs is called *debugging*
- *Syntax error:* A grammatical mistake in a program
  - The compiler can detect these errors, and will output an error message saying what it thinks the error is, and where it thinks the error is

Tip: Error Messages

- *Run-time error:* An error that is not detected until a program is run
  - The compiler cannot detect these errors: an error message is not generated after compilation, but after execution
- *Logic error:* A mistake in the underlying algorithm for a program
  - *The compiler cannot detect these errors, and no error message is generated after compilation or execution, but the program does not do what it is supposed to do*

Naming Conventions

- Start the names of variables, classes, methods, and objects with a lowercase letter, indicate "word" boundaries with an uppercase letter, and restrict the remaining characters to digits and lowercase letters

  `topSpeed` `bankRate1` `timeOfArrival`

- Start the names of classes with an uppercase letter and, otherwise, adhere to the rules above

  `FirstProgram` `MyClass` `String`