Java Programming, Comprehensive

Lecture 1

Bineet Sharma

Material adopted from Murach's © book: Do not distribute!

What Are We Doing Today?

- Class logistics
- Introducing ourselves
- About Java
  - History of Java
  - Benefits of Java
  - JDK
  - JVM
- Recap of Java
- Hands-on Exercise: Eclipse

You (Yes, You!) Should Already…

- Understand a basic/introductory level of Java language
- Have the Java development environment in your computer
  - Verify your JDK and IDE are installed properly
- Know how to create projects, import external projects and files into Eclipse projects

What Do We Do In This Class, Anyway?

- We meet 10 times
  - We have no class on 9/2
- We learn/teach/lecture/drink free coffee for 3 hours
- We work on…
  - 2 Programming Assignments
  - 4 Quizzes
  - Thousands (at least 5) of Homework Assignments
- We get graded (your grade preference choice on 1st lecture & 9th lecture – You can have letter grade – by default, also either P/NP, or NC)
- We never stop talking to each other (portal, message boards, LinkedIn)

What Do We Do In This Class, Anyway?

- We follow Murach's Java Programming, 4th ed., religiously
  - Also refer Java The Complete Reference, 8th Edition, by Herbert Schildt
  - We occasionally look at Introduction to Java Programming by Y. Daniel Liang, 10th Edition
- We never stop consulting Java references and tutorials from Oracle. https://docs.oracle.com/javase/tutorial/java/
- We are skeptical about all the information out there in the internet, except some good ones, including sites:*.edu
- If rusted, come to speed, use: "Practice Your Java Level 1"
  - https://www.amazon.com/gp/product/B01A3XL32W/ref=dp-kindle-redirect?ie=UTF8&btkr=1
- If you had to pick one book then Murach's is highly recommended – Liang's is my other favorite

How You Are Getting Evaluated:

- 40% = 4 Quizzes
- 55% = 2 Big Programming Projects (20%+35%)
- 5% = 5 out of 8 HW Assignments
  - Homework are available in assignment section
  - Submit to get two rewards:
    - Part of 5% of your total grade
    - Model answer
- Class Participation is Encouraged
  - Attendance
  - Code reviews
  - Q & A in forum

Are You in the Right Class?

- <u>What will we cover?:</u>
  - Fast paced Java language overview, good coding practice, OOP, XML, GUI, Collections & Generics, Input/Output Programming, Network Programming, and Multithreading

- <u>Who will benefit most?:</u>
  - If you have taken introductory Java classes
  - Gone through lecture notes every week
  - Complete test your understanding quiz, homework assignment in a weekly basis

- <u>Who will have difficulty?:</u>
  - If you are brand new to Java and/or play the catch-up

What Are We Doing Today?

- Welcome – let us introduce ourselves
- Who am I?
  - Software Engineer with 25+ years of programming experience
  - Instructor at various UC & Silicon valley campuses
  - Decades of teaching experience
  - Leading a cool startup in mobile platform
- Who are you?
  - Your programming experience
  - Your professional background
  - Your expectation out of this class

Why Java?

- Because, Java is:
  - Simple:
    - General purpose, high level programming language
    - More functionality and fewer negatives
    - No pointers, automatic garbage collection, rich pre-defined class libraries
  - Object Oriented:
    - Focuses on objects state and behaviors. Supports Object Oriented Programming (OOP) from the ground up
    - No code outside of a class, no global variables
    - Greater flexibility, modularity, clarity, and reusability - achieved through encapsulation, inheritance and polymorphism

Why Java?

- Because, Java is:
  - Interpreted:
    - Byte codes: compiler generates platform independent Java Virtual Machine (JVM) (instead of native machine) code like other languages
    - Interpreted by a interpreter, which is part of JVM (a java run time), specific to that machine
    - JVM: a self-contained platform runtime system for diverse OS hardware
  - Portable:
    - Architecture neutral, true write once, run everywhere
    - JVM allows same application to run on all operating system
    - The sizes of primitive data types are same

Why Java? (continued)

- Because, Java is:
  - Robust:
    - Excellent compile-time and run-time error checking and exception handling allowing programmers to eliminate buggy code
    - No pointers, so, no memory leak issues
    - No worries about un-authorized access of memory
    - Memory not used (garbage) is collected automatically
  - Secure:
    - Access restrictions are forced
    - Memory allocation model comes to the defense for the networked environments vulnerability

Why Java? (continued)

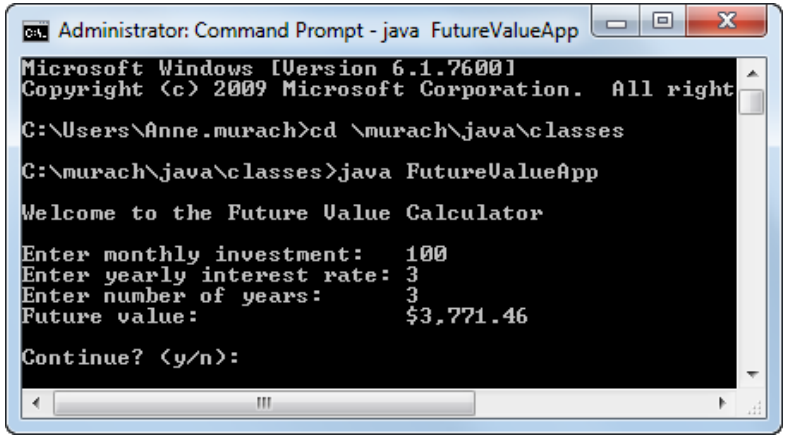
- Because, Java is:
  - Multithreaded:
    - You can run multiple snippet of the codes concurrently using threads
    - Java has simple, easy to use yet robust multithreading primitives embedded in language rather than OS specific procedure calls
  - Dynamic:
    - Java adapts with changes in environment
    - Libraries are constantly upgraded without breaking the old codes
    - Interface concept allows the future expansion
    - Run time type checking is very versatile
    - New features can be incorporated transparently as needed

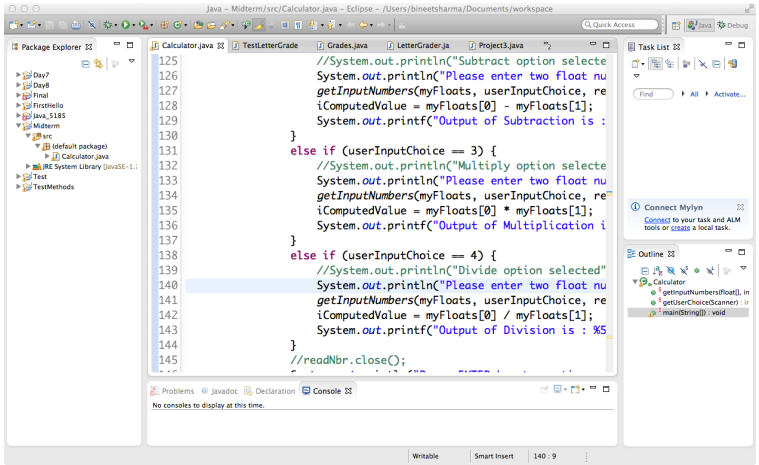
Why Java? (continued)

- Distributed:
  - Java makes distributed computing (several computer working together over the network) seamless as if you are working with a file locally
- It is inspired by C/C++ and Smalltalk
  - At times called (C++--)
- Is Java Slow?
  - It is slower than compiled languages
  - However, there are recent advances in JVM technologies, e.g., Just In Time (JIT) compiler

What Can You Do With Java?

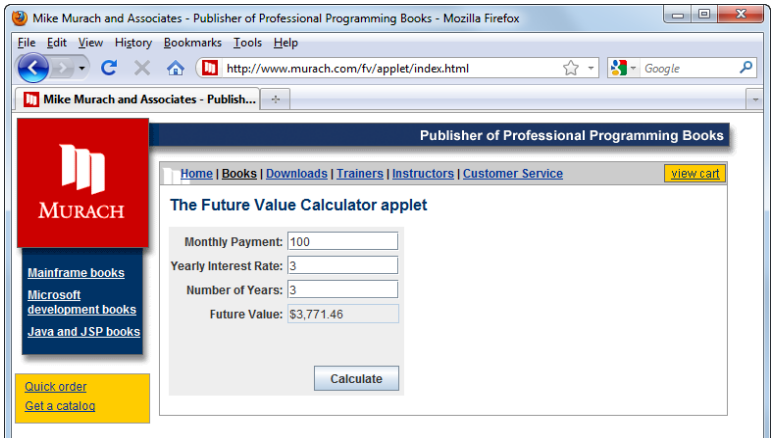
- Stand-alone Applications:
  - Simple console applications
  - Window based desktop applications using JFX, Swing, AWT
    - Eclipse is developed using Java

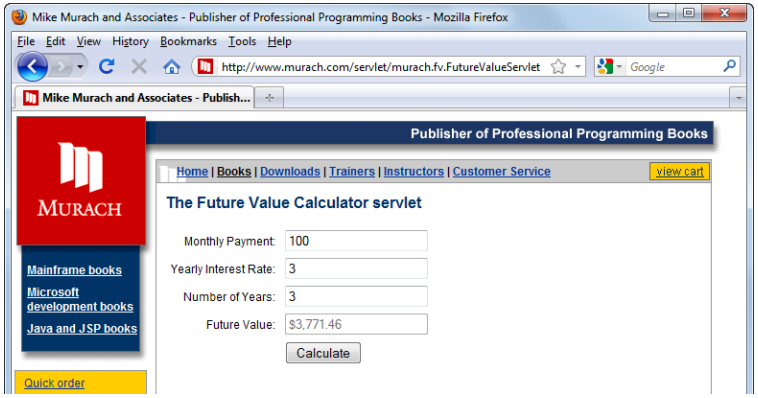




What Can You Do With Java?

- Web Applications:
  - Applets (less and less common nowadays)
  - Servlets, JSPs
- Server Side Applications:
  - Database programming using JDBC
  - Network programming
  - Web Services
- Mobile Applications:
  - Popular in Android platform





Is There Any Application You Can't Write with Java?

- Device drivers
  - Low level OS calls are available but only through native extensions, which are not in Java
- Real-Time applications
  - Java speed is improving, this is not an issue in most cases
- Unsupported devices
  - E.g. iPhone app development

Java History & Timeline:

- James Gosling, Main Architect
- Dec., 1990 Internal Project
  - Internal green project in Sun Microsystems
  - Initially designed for set-top boxes – called OAK
- January, 1996 JDK 1.0
- August, 1999 J2SE
- Dec., 1999 J3EE
- April, 2010 Oracle Buys Sun
- July, 2011 Java SE 7 (1.7) – We will stay with this release
- March 18, 2014 Java SE 8
- More at http://en.wikipedia.org/wiki/Java_version_history

Java History

- Java Development Kit (JDK) is an extended subset of a Software Development Kit (SDK).
  - It includes tools for developing, debugging, and monitoring Java applications
- JDK & SDK are interchangeably used
- Java 8.0 means JDK 1.8
- JDK 1.2 to 1.5
  - Standard Edition (SE) was called Java 2 Platform, Standard Edition (J2SE)
  - Enterprise Edition (EE) was called Java 2 Platform, Enterprise Edition (J2EE)
- After 1.6
  - Standard Edition (SE) is called Java SE
    - Currently Java SE 8 JDK (released on March 2014)
  - Enterprise Edition (EE) is called Java EE

What Is Java?

- Java consists of three important components:
  - Java Language:
    - Defines the syntax and semantics of the Java programming language
  - The Java Virtual Machine (JVM)
    - Is a platform – has tools to run your code in a native operating system
  - The Java API (Application Programming Interface)
    - Has a vast range of library routines (API) for GUI, data storage, processing, I/O, and networking
- Java has a complete set of application development tools
- Java is not JavaScript!

Java Virtual Machine (JVM)

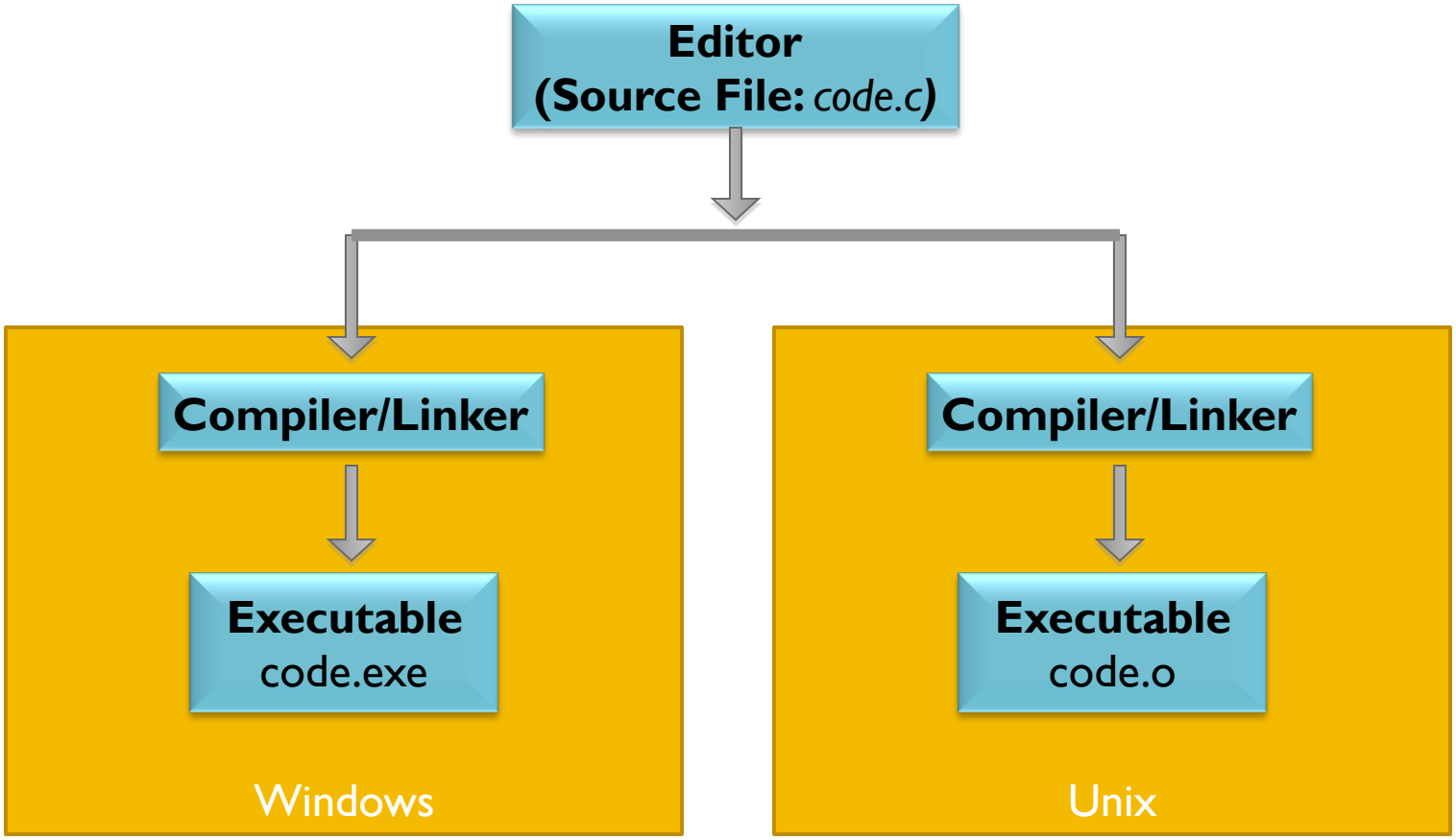
- JVM is a platform – a run-time environment
  - It is like an OS inside an OS
  - Most operating systems have JVM:
    - Windows, Unix, Linux, Mac
  - It allows you to run a Java stand alone application inside OS
  - It is embedded in most browsers as well; this allows you to run Java applications inside the browser (Java Applets)

Java Virtual Machine (JVM) (cont.)

- How does JVM work?
  - Java code (.java files) are compiled to byte-code (for JVM, not for target OS)
  - The embedded JVM in target OS will verify byte-code, using verifier
  - Interpreter within the JVM will interpret the code, and ultimately JVM will execute the code in target OS

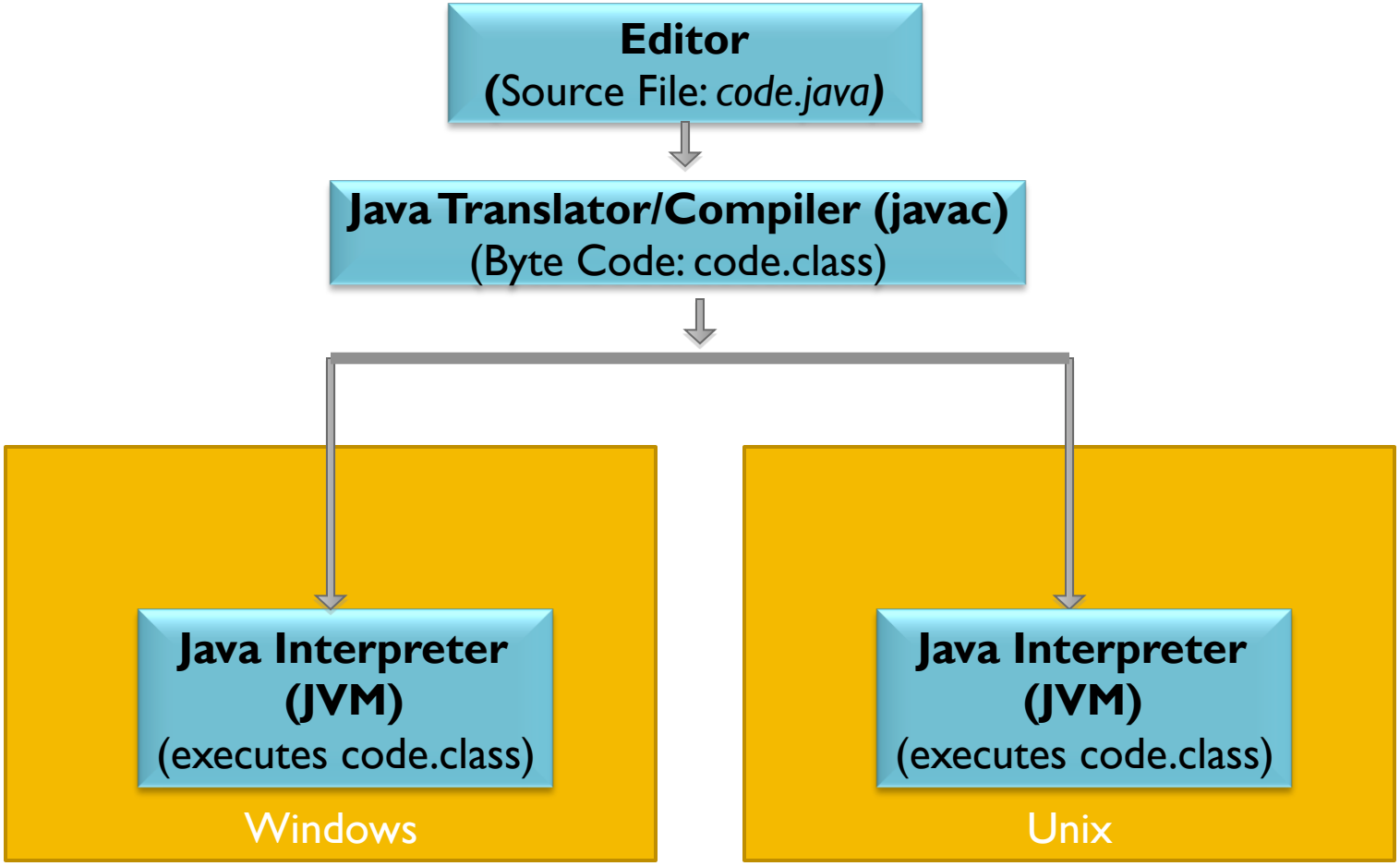
Java Virtual Machine (JVM)

- How do other languages compile and execute code?



Java Virtual Machine (JVM)

- How does Java compile and execute code?



Java Virtual Machine (JVM)

- Advantages of JVM
  - Translated (compiled) code (byte code) runs in any operating system that has a suitable JVM without any modification
  - True **compile once, run anywhere**
- Disadvantages of JVM
  - Since it is interpreted during execution, it will run slower than a fully compiled code; however, JVM is getting smarter by each revision; Just In Time (JIT) compiler speeds things up
  - Low level device manipulation is not available in JVM, and it needs to rely on native extensions provided by the host environment (Operating System)

A Language of Choice - Benefits

- Up and running fast - easy to learn!
- Write less code
- Forces you to write better code: OOP, API, GC helps
- Release your project early
- No worries about OS – same data types for all OS
- WORA: Write Once, Run Anywhere
- New releases are a breeze
- Wide support for rich GUI, Web, Network, Database, Mobile application developments
- Many user groups, knowledge share, open source

Ready To Get Started? Murach's Chapter 01 FutureValueApp.Java

```
Enter monthly investment:   100
Enter yearly interest rate: 12
Enter number of years:      30
Future value:               $352,991.38

Continue? (y/n): y

Enter monthly investment:   100
Enter yearly interest rate: 8
Enter number of years:      60
Future value:               $1,790,711.77

Continue? (y/n): n
```

Ready To Get Started? Murach's Chapter 01 FutureValueApp.Java

The code for the Future Value application

```
import java.util.Scanner;
import java.text.NumberFormat;

public class FutureValueApp
{
    public static void main(String[] args)
    {
        System.out.println(
            "\nWelcome to the Future Value Calculator\n");

        Scanner sc = new Scanner(System.in);
        String choice = "y";

        while (choice.equalsIgnoreCase("y"))
        {
            // get the input from the user
            System.out.print(
                "Enter monthly investment:   ");
            double monthlyInvestment = sc.nextDouble();
            System.out.print(
                "Enter yearly interest rate: ");
```

Ready To Get Started? Murach's Chapter 01 InvoiceApp.Java

The code for the Future Value application (cont.)

```
double interestRate = sc.nextDouble();
System.out.print(
    "Enter number of years:      ");
int years = sc.nextInt();

// calculate the future value
double monthlyInterestRate =
    interestRate/12/100;
int months = years * 12;
double futureValue = calculateFutureValue(
    monthlyInvestment, monthlyInterestRate,
    months);

// format and display the result
NumberFormat currency =
     NumberFormat.getCurrencyInstance();
System.out.println(
     "Future value:                " +
     currency.format(futureValue) + "\n");
```

Ready To Get Started? Murach's Chapter 01 InvoiceApp.Java

The code for the Future Value application (cont.)

```
            // see if the user wants to continue
            System.out.print("Continue? (y/n): ");
            choice = sc.next();
            System.out.println();
        }
    }

    private static double calculateFutureValue(
             double monthlyInvestment,
             double monthlyInterestRate, int months)
    {
        double futureValue = 0;
        for (int i = 1; i <= months; i++)
            futureValue =
            (futureValue + monthlyInvestment) *
            (1 + monthlyInterestRate);
        return futureValue;
    }
}
```

Java Overview

- All Java codes are written within some classes
- No global functions or variables concept (outside of class)
- A Java application contains hundreds of classes if not more
- An application has at least one public class. Each public class is saved in a disk file of same name (with .java ext.)
- One Java file can contain additional classes, which are not public (private or package visible classes are fine)
- A Java class consists of fields (variables) and methods
  - block of code which does something, like functions in other language
- A block of code is written within curly braces { }

Java Overview

- Class names:

The rules for naming a class

- Start the name with a capital letter.
- Use letters and digits only.
- Follow the other rules for naming an identifier.

Recommendations for naming a class

- Start every word within a class name with an initial cap.
- Each class name should be a noun or a noun that's preceded by one or more adjectives.
    - For example: *RoundBox, WesternState, ComplexNumber*

Java Overview

- A Java program consists of:
  - Comments
    - Three types of comments
      - Line comment:

```
int dogAge;          //store age of the dog
```

      - Paragraph comment, can be used between codes

```
/* This is a program which simulates
dog's behavior */

int /*store age of the dog*/ dogAge;
```

Java Overview

- A Java program consists of:
  - Comments
    - Three types of comments
    - …
      - Javadoc comment is used for documenting:
        - Classes
        - Data
        - Methods
        - Interfaces

```
/** This is a class which
 simulates dog's behavior */

class Dog { }
```

      - It can be extracted into an HTML document using JDK's javadoc command for a professional looking help file

Java Overview

- A Java program consists of:
  - Reserved/Key Words
    - Have a specific meaning to the compiler and cannot be used for naming your identifiers (classes, methods and variables)

      For example: class, static, void, public etc.

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

Java Overview

- A Java program consists of:
  - Reserved/Key Words
  - What is not a keyword?
    - main, System, out, println, Scanner, choice, "y" are not keywords
    - What are they?

The code for the Future Value application

```
import java.util.Scanner;
import java.text.NumberFormat;

public class FutureValueApp
{
    public static void main(String[] args)
    {
        System.out.println(
            "\nWelcome to the Future Value Calculator\n");

        Scanner sc = new Scanner(System.in);
        String choice = "y";

        while (choice.equalsIgnoreCase("y"))
        {
            // get the input from the user
            System.out.print(
                "Enter monthly investment:   ");
            double monthlyInvestment = sc.nextDouble();
            System.out.print(
                "Enter yearly interest rate: ");
```

Java Overview

- A Java program consists of:
  - Identifier: given name to your data items

Valid identifiers

```
InvoiceApp          $orderTotal      i
Invoice             _orderTotal      x
InvoiceApp2         input_string     TITLE
subtotal            _get_total       MONTHS_PER_YEAR
discountPercent     $_64_Valid
```

The rules for naming an identifier

- Start each identifier with a letter, underscore, or dollar sign. Use letters, dollar signs, underscores, or digits for subsequent characters.
- Use up to 255 characters.
- Don't use Java keywords.

Java Overview

- A Java program consists of: Variables

Naming recommendations for variables

- Start variable names with a lowercase letter and capitalize the first letter in all words after the first word.
- Each variable name should be a noun or a noun preceded by one or more adjectives.
- Try to use meaningful names that are easy to remember.

Java Overview

- A Java program consists of
  - Expressions: combinations of operators and operands, and method calls
  - Statements: expressions, assignments which are complete instruction to do some action. Statements complete with a semicolon (;)

Statements that use simple arithmetic expressions

```
// integer arithmetic
int x = 14;
int y = 8;
int result1 = x + y;          // result1 = 22
int result2 = x - y;          // result2 = 6
int result3 = x * y;          // result3 = 112
int result4 = x / y;          // result4 = 1

// double arithmetic
double a = 8.5;
double b = 3.4;
double result5 = a + b;       // result5 = 11.9
double result6 = a - b;       // result6 = 5.1
double result7= a * b;        // result7 = 28.9
double result8 = a / b;       // result8 = 2.5
```

Java Overview

- You typically work with numbers and strings

Statements that mix int and double variables

```
int result9 = invoiceTotal / invoiceCount
                                    // result9 = 125
double result10 = invoiceTotal / invoiceCount
                                    // result10 = 125.50

Note: say double invoiceTotal = 251.0, and
          int invoiceCount = 2
```

The syntax for declaring and initializing a string variable

```
String variableName = value;
```

Statements that declare and initialize a string

```
String message1 = "Invalid data entry.";
String message2 = "";
String message3 = null;
```

Java Overview

- Working with strings:

Join strings

```
String firstName = "Bob";              // firstName is Bob
String lastName = "Smith";             // lastName is Smith
String name = firstName + " " + lastName;
                                       // name is Bob Smith
```

Join a string and a number

```
double price = 14.95;
String priceString = "Price: " + price;
```

Java Overview

- Working with strings:

How to append one string to another with the + operator

```
firstName = "Bob";      // firstName is Bob
lastName = "Smith";     // lastName is Smith
name = firstName + " ";
                        // name is Bob followed by a space
name = name + lastName;
                        // name is Bob Smith
```

How to append one string to another with the += operator

```
firstName = "Bob";      // firstName is Bob
lastName = "Smith";     // lastName is Smith
name = firstName + " ";
                        // name is Bob followed by a space
name += lastName;       // name is Bob Smith
```

Java Overview

- A Java program consists of Methods
  - Collection of statements which collectively performs a task. It works like a black box

Static method

```
{
…     // more code
…
    // a static method that requires three arguments
    // and returns a double
    private static double calculateFutureValue(
              double monthlyInvestment,
              double monthlyInterestRate,
              int months)                        {

        double futureValue = 0.0;
        for (int i = 1; i <= months; i++) {
            futureValue =
                (futureValue + monthlyInvestment) *
                (1 + monthlyInterestRate);
        }
        return futureValue;
    }
}
```

Java Overview

- A Java class usually consist of a main method

A public class that contains a main method

```
public class InvoiceApp                    // declare the class
{                                          // begin the class
    public static void main(String[] args)
    {
        System.out.println(
            "Welcome to the Invoice Total Calculator");
    }
}                                          // end the class
```

```
Welcome to the Invoice Total Calculator

Enter subtotal:   150
Discount percent: 0.1
Discount amount:  15.0
Invoice total:    135.0

Continue? (y/n): y
```

Java Overview

- A Java class can consist of a main method (cont.)
  - main() is a special method – a class can contain a main method
  - If you want JVM to execute your class, you must provide a main() method
  - main() is the entry point for JVM into your class
  - You typically do not call main() method from within your class in lieu of a loop (doable, but a bad idea)
- One Java application can have multiple main() methods in different classes
- Only one of them is an entry to the application

Java Overview

- A Java program consists of variables and references (objects)

Create an object from a class

Syntax

```
ClassName objectName = new ClassName(arguments);
```

Examples

```
Scanner sc = new Scanner(System.in);
                    // creates a Scanner object named sc
Date now = new Date();
                    // creates a Date object named now
```

Call a method from an object

Syntax

```
objectName.methodName(arguments)
```

Examples

```
double subtotal = sc.nextDouble();
                   // get a double entry from the console
String currentDate = now.toString();
                   // convert the date to a string
```

- What is variable, object, & reference?

Java Overview

- A Java program consists of variables and references (objects)

How to call a static method from a class

Syntax

```
ClassName.methodName(arguments)
```

Examples

```
String sPrice = Double.toString(price);
                              // convert a double to a string
double total = Double.parseDouble(userEntry);
                              // convert a string to a double
```

Java Overview

- A Java program groups related classes into a package

Common library packages

- `java.lang`
- `java.text`
- `java.util`
- `java.io`
- `java.sql`
- `java.applet`
- `java.awt`
- `java.awt.event`
- `javax.swing`

The syntax of the import statement

```
import packagename.ClassName;
   or
import packagename.*;
```

Examples

```
import java.text.NumberFormat;
import java.util.Scanner;
import java.util.*;
import javax.swing.*;
```

Java Overview

- A Java program groups related classes into a package
- How to use Scanner class included in util package?

With an import statement

```
import java.util.Scanner;
Scanner sc = new Scanner(System.in);
```

Without an import statement

```
java.util.Scanner sc = new java.util.Scanner(System.in);
```

Java Overview

- A Java program has input and output statements
  - Scanner object is used to get the input from users

Common methods of a Scanner object

- `next()`
- `nextInt()`
- `nextDouble()`
- `nextLine()`

Using the methods of a Scanner object

```
String name = sc.next();
int count = sc.nextInt();
double subtotal = sc.nextDouble();
String cityName = sc.nextLine();
```

- The Scanner class was introduced in version 1.5 of the JDK.

Java Overview

- A Java program has input and output statements

Code that gets three values from the user

```
// create a Scanner object
Scanner sc = new Scanner(System.in);

// read a string
System.out.print("Enter product code: ");
String productCode = sc.next();

// read a double value
System.out.print("Enter price: ");
double price = sc.nextDouble();

// read an int value
System.out.print("Enter quantity: ");
int quantity = sc.nextInt();

// perform a calculation and display the result
double total = price * quantity;
System.out.println();
System.out.println(quantity + " " + productCode
    + " @ " + price + " = " + total);
System.out.println();
```

Java Overview

- A Java program has input and output statements

Code that reads three values from one line

```
// read three int values
System.out.print("Enter three integer values: ");
int i1 = sc.nextInt();
int i2 = sc.nextInt();
int i3 = sc.nextInt();

// calculate the average and display the result
int total = i1 + i2 + i3;
int avg = total / 3;
System.out.println("Average: " + avg);
System.out.println();
```

The console after the program finishes

```
Enter three integer values: 99 88 92
Average: 93
```

Java Overview

- A Java program has 8 different basic data types (primitive)
  - More on data types: http://docs.oracle.com/javase/tutorial/java/nutsandbolts/datatypes.html

The eight primitive data types

Type	Bytes	Use
byte	1	Very short integers from -128 to 127.
short	2	Short integers from -32,768 to 32,767.
int	4	Integers from -2,147,483,648 to 2,147,483,647.
long	8	Long integers from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807.
float	4	Single-precision, floating-point numbers from -3.4E38 to 3.4E38 with up to 7 significant digits.
double	8	Double-precision, floating-point numbers from -1.7E308 to 1.7E308 with up to 16 significant digits.
char	2	A single Unicode character that's stored in two bytes.
boolean	1	A *true* or *false* value.

Java Overview

Working with floating point numbers:

- To express the value of a floating-point number you can use *scientific notation* like:
  - 2.382E+5which means 2.382 times 10^5
    - a value of 238,200, or
  - 3.25E-8, which means 3.25 times 10^{-8}
    - a value of .0000000325
  - Java will sometimes use this notation to display the value of a float or double data type.
- Because of the way floating-point numbers are stored internally, they can't represent the exact value of the decimal places in some numbers.
  - This can cause a rounding problem in some business applications.

Java Overview

- Declare and initialize variables using two statements

Syntax

```
type variableName;
variableName = value;
```

Example

```
int counter;                    // declaration statement
counter = 1;                    // assignment statement
```

Java Overview

- Declare and initialize variables using one statement

Syntax

```
type variableName = value;
```

Examples

```
int counter = 1;        // initialize an int variable
                        // 1 is a literal - constant
double price = 14.95; // initialize a double variable
float interestRate = 8.125F;
// F indicates a floating-point value, why F is needed?
long numberOfBytes = 20000000000L;
// L indicates a long integer, why L is needed?
int population = 1_734_323;
// underscores improve readability, interesting…
double distance = 3.65e+9; // scientific notation
char letter = 'A';
                // stored as a two-digit Unicode character
char letter = 65;
                // integer value for a Unicode character
boolean valid = false; // where false is a keyword
int x = 0, y = 0; // 2 variables with 1 statement
```

Java Overview

- Variable naming conventions:
  - Start variable names with a lowercase letter and capitalize the first letter in all words after the first word.
  - Try to use meaningful names that are easy to remember as you code.
  - The use of underscores (1_734_123) in numeric literals was introduced in version 1.7 of the JDK.

Java Overview

- Working with constants:

Syntax

```
final type CONSTANT_NAME = value;
```

Examples

```
final int DAYS_IN_NOVEMBER = 30;
final float SALES_TAX = .075F;
final double LIGHT_YEAR_MILES = 5.879e+12;
```

- Constant naming conventions:

  - Capitalize all of the letters in constants and separate words with underscores.
  - Try to use meaningful names that are easy to remember.

Java Overview

- Arithmetic Operators

Operator	Name
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulus
++	Increment
--	Decrement
+	Positive sign
-	Negative sign

Java Overview

- Examples of simple assignment statements

```
int x = 14, y = 8;
int result1 = x + y;        // result1 = 22
int result2 = x - y;        // result2 = 6
int result3 = x * y;        // result3 = 112
int result4 = x / y;        // result4 = 1
int result5 = x % y;        // result5 = 6
int result6 = -y + x;       // result6 = 6
int result7 = --y;          // result7 = 7, y = 7
int result8 = ++x;          // result8 = 15, x = 15

double a = 8.5;
double b = 3.4;
double result9 = a + b;     // result9 = 11.9
double result10 = a - b;    // result10 = 5.1
double result11 = a * b;    // result11 = 28.90
double result12 = a / b;    // result12 = 2.5
double result13 = a % b;    // result13 = 1.7
double result14 = -a + b;   // result14 = -5.1
double result15 = --a;      // result15 = 7.5
double result16 = ++b;      // result16 = 4.4

char letter1 = 'C';
char letter2 = ++letter1;
```

Java Overview

- Assignment Operators

Operator	Name
`=`	Assignment
`+=`	Addition
`-=`	Subtraction
`*=`	Multiplication
`/=`	Division
`%=`	Modulus

Java Overview

- Assignment Operators: Example – same variable on both sides of equal sign

```
count = count + 1;          // count is increased by 1
count = count - 1;          // count is decreased by 1
total = total + 100.0;      // total is increased by 100.0
total = total - 100.0;      // total is decreased by 100.0
price = price * .8;         // price is multiplied by .8
sum = sum + nextNumber;     // sum is increased by value
                            // of nextNumber
```

- Use shortcut operators instead – you get same result

```
count += 1;            // count is increased by 1
count -= 1;            // count is decreased by 1
total += 100.0;        // total is increased by 100.0
total -= 100.0;        // total is decreased by 100.0
price *= .8;           // price is multipled by .8
sum += nextNumber;     // sum is increased by the value
                       // of nextNumber
```

Java Overview

- The order of precedence for arithmetic operations

The order of precedence for arithmetic operations

1. Increment and decrement
2. Positive and negative
3. Multiplication, division, and remainder
4. Addition and subtraction

- Examples:

Using the default order of precedence

```
double discountPercent = .2;              // 20% discount
double price = 100;                       // $100 price
price = price * 1 - discountPercent;      // price = $99.8
```

Using parentheses that specify the order of precedence

```
price = price * (1 - discountPercent);  // price = $80
```

Java Overview

- The order of precedence for arithmetic operations
- More examples:

Using parentheses that specify the order of precedence

```
double currentValue = 5000;
                    // current value of investment account
double monthlyInvestment = 100;
                    // amount added each month
double yearlyInterestRate = .12;
                    // yearly interest rate

currentValue = (currentValue + monthlyInvestment) *
               (1 + (yearlyInterestRate/12));
```

Without using parentheses

```
currentValue += monthlyInvestment;      // add investment
double monthlyInterestRate = yearlyInterestRate / 12;
double monthlyInterest =
    currentValue * monthlyInterestRate;
currentValue += monthlyInterest;        // add interest
```

Will this produce correct result? Is this better way?

Java Overview

- Prefixed and postfixed increment and decrement operators, examples:

```
int a = 5;
int b = 5;
int y = ++a;      // a = 6, y = 6
int z = b++;      // b = 6, z = 5

//will these even compile? What will y and a have?
//assume a is 5 at the beginning of each statement
y = +a++;
y = -a++;
y = -++a;
y = +++a;
y = +--a;
y = ---a;
```

Java Overview

- Working with different data types in same statements
  - Java allows implicit and explicit type casting to make them compatible
  - Here is how implicit casting works:

Casting from less precise to more precise data types

byte→short→int→long→float→double

Called widening conversion

Examples

```
double grade = 93;     // convert int to double

double d = 95.0;
int i = 86, j = 91;
double average = (d+i+j)/3;
                       // convert i and j to double values
                       // average = 90.666666...
```

Java Overview

- Explicit casting: you force the casting in the code. Why?

Syntax

```
(type) expression
```

Called narrowing conversion

Examples

```
int grade = (int) 93.75;
                 // convert double to int (grade = 93)

double d = 95.0;
int i = 86, j = 91;
double average = ((int)d+i+j)/3;
                 // convert d to int value (average = 90)

double result = (double) i / (double) j;
                 // result has decimal places
char letterChar = 65;
                // convert int to char (letterChar = 'A')
char letterChar2 = (char) 65;
                // this works too

int letterInt = 'A';
```

Java Overview

- Formatting your data. Use NumberFormat class, and use the static methods using class and/or normal methods using objects

The NumberFormat class:

```
java.text.NumberFormat
```

Three static methods of the NumberFormat class, returns an instance of NumberFormat:

- `getCurrencyInstance()`
- `getPercentInstance()`
- `getNumberInstance()`

Three methods of a NumberFormat object:

- `format(anyNumberType)`
- `setMinimumFractionDigits(int)`
- `setMaximumFractionDigits(int)`

Java Overview

- Formatting your data. Examples:

The number format with one decimal place:

```
double miles = 15341.253;
NumberFormat number = NumberFormat.getNumberInstance();
number.setMaximumFractionDigits(1);
String milesString = number.format(miles);
                                        // returns 15,341.3
```

Two NumberFormat methods coded in one statement:

```
String majorityString =
    NumberFormat.getPercentInstance().format(majority);
```

Java Overview

- Java makes math easy! Use the Math class

The Math class

```
java.lang.Math
```

Common static methods of the Math class

- `round(floatOrDouble)`
- `pow(number, power)`
- `sqrt(number)`
- `max(a, b)`
- `min(a, b)`
- `random()`

Java Overview

- Use static methods of Math class

The round method

```
long result = Math.round(1.667);      // result is 2
int result = Math.round(1.49F);       // result is 1
```

The pow method

```
double result = Math.pow(2, 2);
                              // result is 4.0 (2*2)
double result = Math.pow(2, 3);
                              // result is 8.0 (2*2*2)
double result = Math.pow(5, 2);
                              // result is 25.0 (5 squared)
int result = (int) Math.pow(5, 2);
                              // result is 25 (5 squared)
```

The sqrt method

```
double result = Math.sqrt(20.25);     // result is 4.5
```

Java Overview

- Java provides classes like Integer, Double to create objects of primitive data types, like int, double, called wrapper classes

Constructors for the Integer and Double classes

- `Integer(int)`
- `Double(double)`

How to create Integer and Double objects

```
Integer quantityIntegerObject = new Integer(quantity);
Double priceDoubleObject = new Double(price);
```

Java Overview

- Working with Integer, and Double classes

Two static methods of the Integer class

- `parseInt(stringName)`
- `toString(intName)`

Two static methods of the Double class

- `parseDouble(stringName)`
- `toString(doubleName)`

How to use static methods to convert primitive types to String objects

```
String counterString = Integer.toString(counter);
String priceString = Double.toString(price);
```

How to use static methods to convert String objects to primitive types

```
int quantity = Integer.parseInt(quantityString);
double price = Double.parseDouble(priceString);
```

Java Overview

- Example: Formatted Invoice application. Output:

```
Enter subtotal:   150.50
Discount percent: 10%
Discount amount:  $15.05
Total before tax: $135.45
Sales tax:        $6.77
Invoice total:    $142.22

Continue? (y/n):
```

Java Overview

- Example: Formatted Invoice application. Code:

The code for the formatted Invoice application

```
import java.util.Scanner;
import java.text.NumberFormat;

public class FormattedInvoiceApp{
    public static void main(String[] args) {

        final double SALES_TAX_PCT = .05;

        Scanner sc = new Scanner(System.in);
        String choice = "y";
        while (choice.equalsIgnoreCase("y")) {
            // get the input from the user
            System.out.print("Enter subtotal:   ");
            double subtotal = sc.nextDouble();

            // calculate the results
            double discountPercent = 0.0;
            if (subtotal >= 100)
                discountPercent = .1;
            else
                discountPercent = 0.0;
```

Java Overview

- Example: Formatted Invoice application. Code (cont.):

```
double discountAmount =
    subtotal * discountPercent;
double totalBeforeTax =
    subtotal - discountAmount;
double salesTax =
    totalBeforeTax * SALES_TAX_PCT;
double total = totalBeforeTax + salesTax;

// format and display the results
NumberFormat currency =
    NumberFormat.getCurrencyInstance();
NumberFormat percent =
    NumberFormat.getPercentInstance();
```

Java Overview

- Example: Formatted Invoice application. Code (cont.):

```
            String message =
                "Discount percent: "
              + percent.format(discountPercent) + "\n"
              + "Discount amount:  "
              + currency.format(discountAmount) + "\n"
              + "Total before tax: "
              + currency.format(totalBeforeTax) + "\n"
              + "Sales tax:        "
              + currency.format(salesTax) + "\n"
              + "Invoice total:    "
              + currency.format(total) + "\n";
            System.out.println(message);

            // see if the user wants to continue
            System.out.print("Continue? (y/n): ");
            choice = sc.next();
            System.out.println();
        }
    }
}
```

Java Overview

- Example: Let us run the same application again, and give different input. What do you observe?

```
Enter subtotal:    100.05
Discount percent: 10%
Discount amount:   $10.01
Total before tax: $90.05
Sales tax:         $4.50
Invoice total:     $94.55
Continue? (y/n):
```

More Information

- JDK: https://www.oracle.com/technetwork/java/javase/downloads/index.html
- Eclipse: https://www.eclipse.org/
- Java History: https://en.wikipedia.org/wiki/Java_version_history
- Java Platform SE 7 Documentation https://docs.oracle.com/javase/7/docs/api/index.html
- Java Platform SE 8 Documentation https://docs.oracle.com/javase/8/docs/api/index.html
- Java Tutorials: https://docs.oracle.com/javase/tutorial/java/

Next Lecture

- Rounding issues
  - BigDecimal
- Control statements
  - Relational/Logical operators
  - if/else, nested ifs, switch
  - Loops
- Static method
- Data validation
  - Syntax error
  - Logical error
  - Run-time error

What Will You Do This Week & Next?

- Go through lecture#1 material and demo codes
- Complete test your understanding quiz
  - Test your understanding quizzes are not graded
- Complete HW#1
  - It is available in the assignment section. Submit your work and you will be awarded with "Model Solution" right away.
  - Homework carry 5% of your grade
- Read ahead lecture#2 notes once available

Recap!

- Introducing ourselves
- About Java
  - History of Java
  - Benefits of Java
  - JDK
  - JVM
- Fast paced recap of Java
- Hands on exercise using Eclipse