Advanced Programming Language

MSc. Nguyen Cao Dat

dat@hcmut.edu.vn

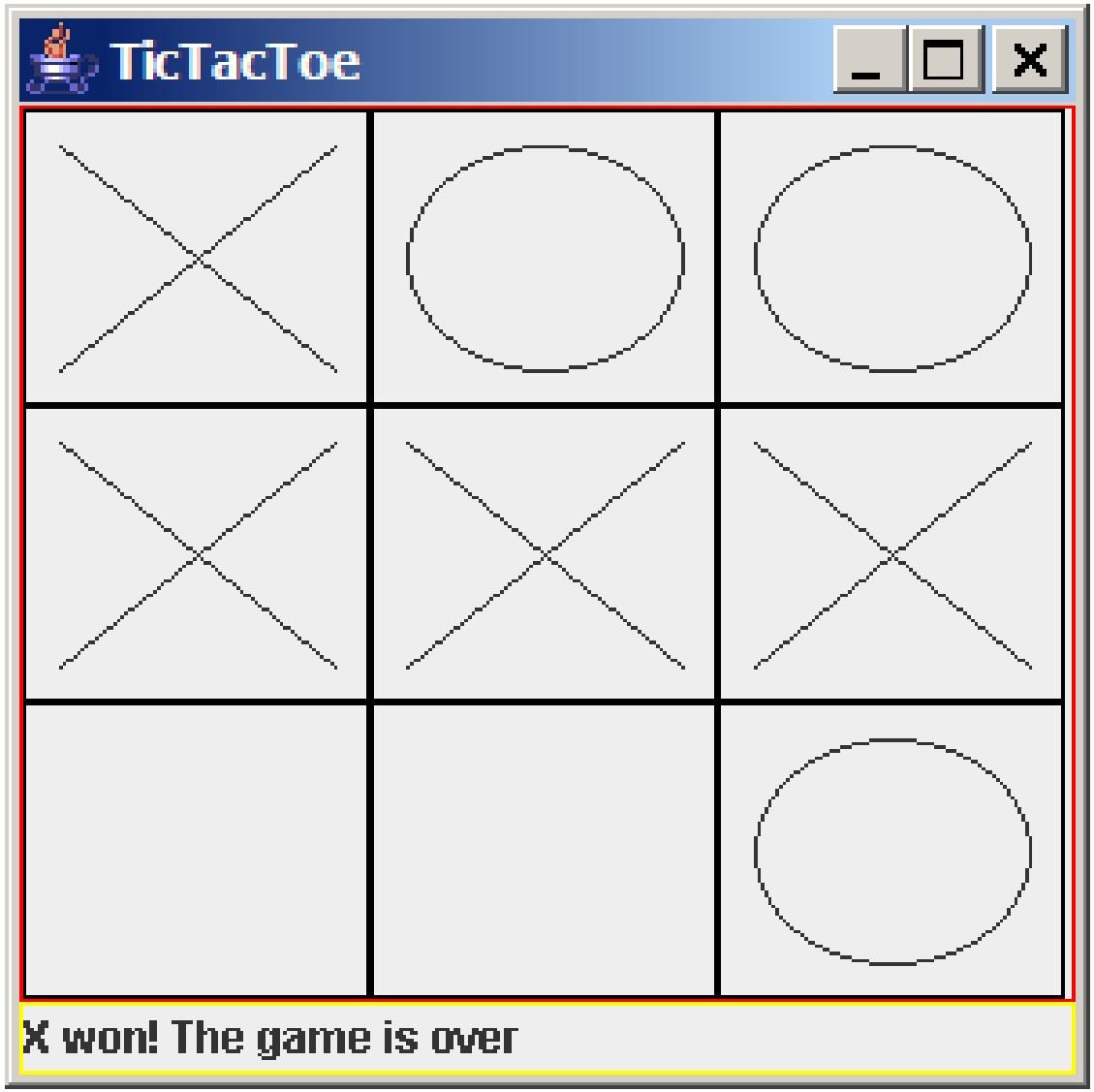
Java Programming Language

- Java is a general purpose programming language.
- Java is the Internet programming language.
- Java can be used to develop Web applications.
- Java can also be used to develop applications for cell phones.

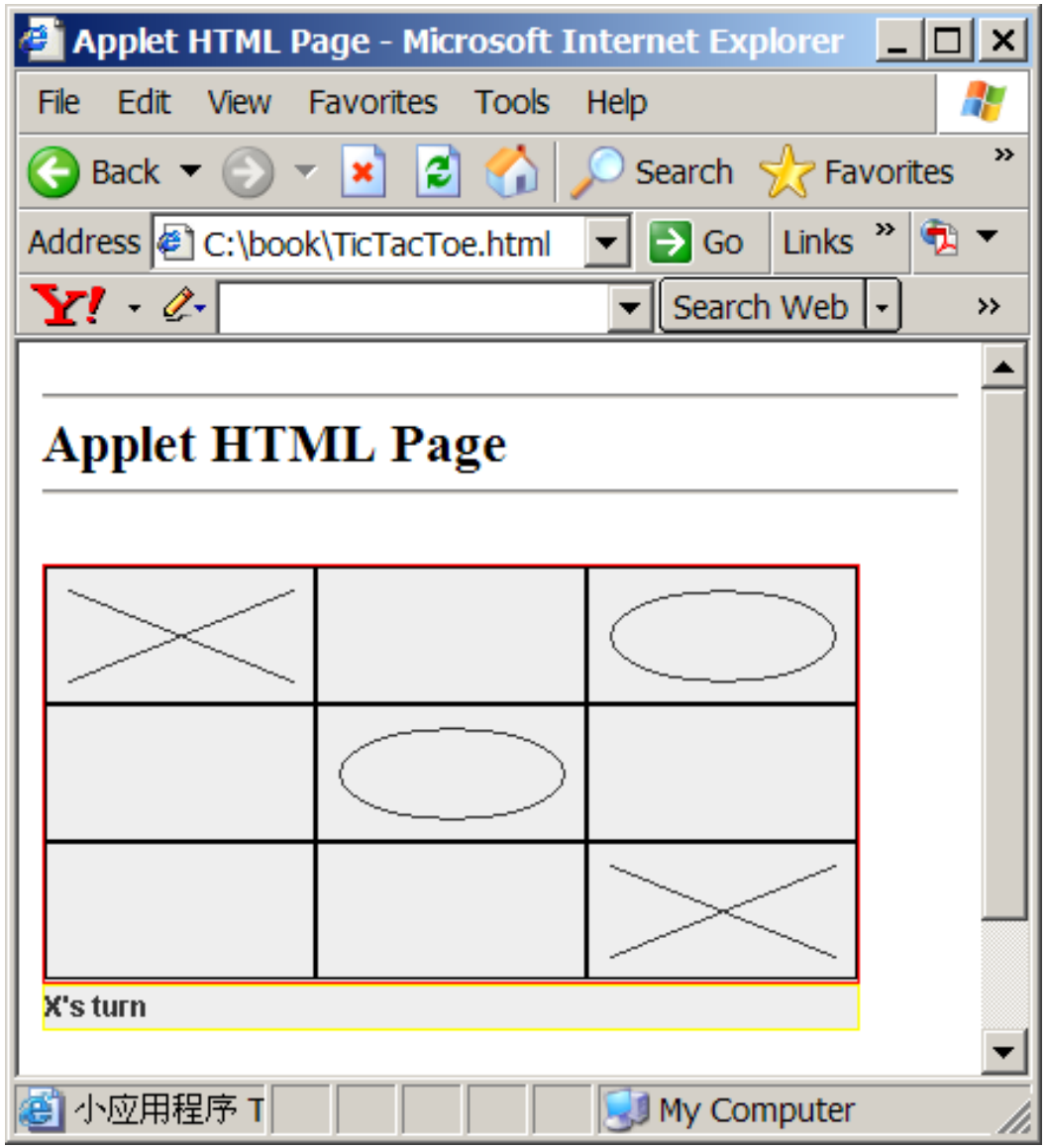
Examples of Java's Versatility

- Standalone Application: TicTacToe
- Applet: TicTacToe
- Servlets: SelfTest Web site
- Mobile Computing: Cell phones

TicTacToe Standalone



TicTacToe Applet



SelfTest Website (using Java Servlets)

http://panda.armstrong.edu:8080/servlet/selftest5e.SelfTest?chapter=1 - Microsoft Internet E...

File Edit View Favorites Tools Help

Back Search Favorites Media

Address http://panda.armstrong.edu:8080/servlet/selftest5e.SelfTest?chapter=1 Go Links

Developed by Y. Daniel Liang using Java Servlets (Current time is Sun Nov 21 20:31:23 EST 2004)

Chapter 1 Introduction to Computers, Programs, and Java

Section 1.2 What is a Computer?

1 _____ is the physical aspect of the computer that can be seen.

- A. Hardware
- B. Software
- C. operating system
- D. Application program

Done Internet

PDA and Cell Phone

Content

- **Module 1**
  - Basic Java Programming
- **Module 2**
  - Java Object Oriented Programming
- **Module 3**
  - Data Structures in Java

Module I

BASIC JAVA PROGRAMMING

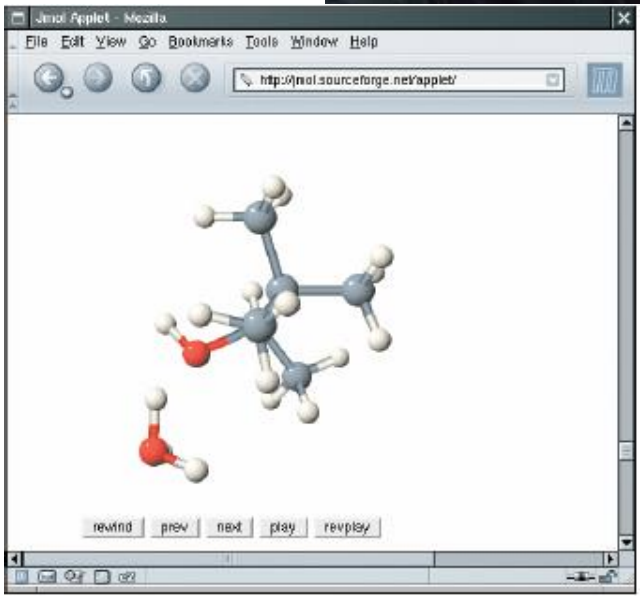
Content

- Components of the Java Environment
- Your First Java Program
- Variables and Primitive Data Types
- Selection Statements
- Loop Statements
- Methods

COMPONENTS OF THE JAVA ENVIRONMENT

The Java Language

- **In 1991, James Gosling of Sun Microsystems designed what would become the Java programming language**
  - In 2010, Sun was acquired by Oracle
- **Platform independent**
  - Can run on almost any machine
- **Used to create Internet applets**



Characteristics of Java

- **Java Is Simple**
- **Java Is Object-Oriented**
- **Java Is Distributed**
- **Java Is Interpreted**
- **Java Is Robust**
- **Java Is Secure**
- **Java Is Architecture-Neutral**
- **Java Is Portable**
- **Java Is Multithreaded**
- **Java Is Dynamic**

JDK(Java Development Kit) Edition

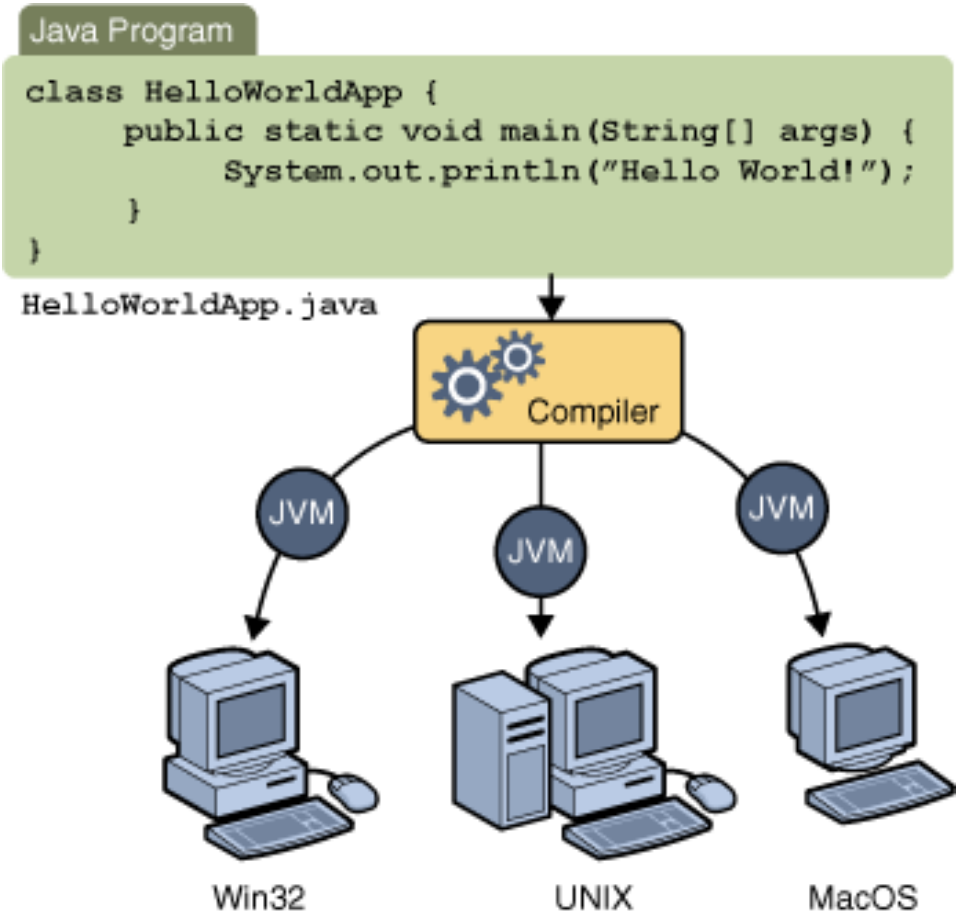
- **Java Standard Edition (J2SE)**
  - J2SE can be used to develop client-side standalone applications or applets.
- **Java Enterprise Edition (J2EE)**
  - J2EE can be used to develop server-side applications such as Java servlets and Java ServerPages.
- **Java Micro Edition (J2ME).**
  - J2ME can be used to develop applications for mobile devices.

This course uses J2SE to introduce Java programming.

JDK Timeline

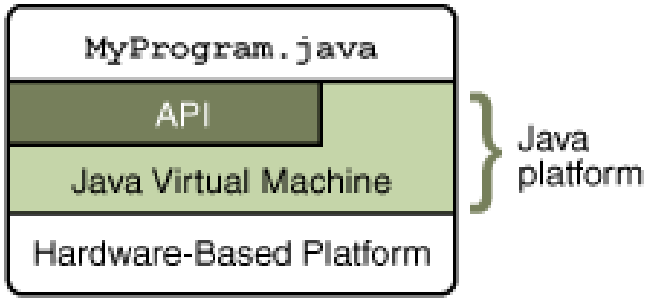
Java Virtual Machine (JVM)

- Java code starts as source code (human-readable)
- A compiler converts it into machine readable code (byte code)
- Any JVM can then run the code, which is in a .class file



Java Platform

- **Consists of two parts**
  - Java Virtual Machine (JVM)
  - Java Application Programming Interface (API)
    - A huge collection of handy software packages that programmers can use
      - Graphics, user interface, networking, sound, database, math, etc.
    - Helps programmers not have to reinvent the wheel



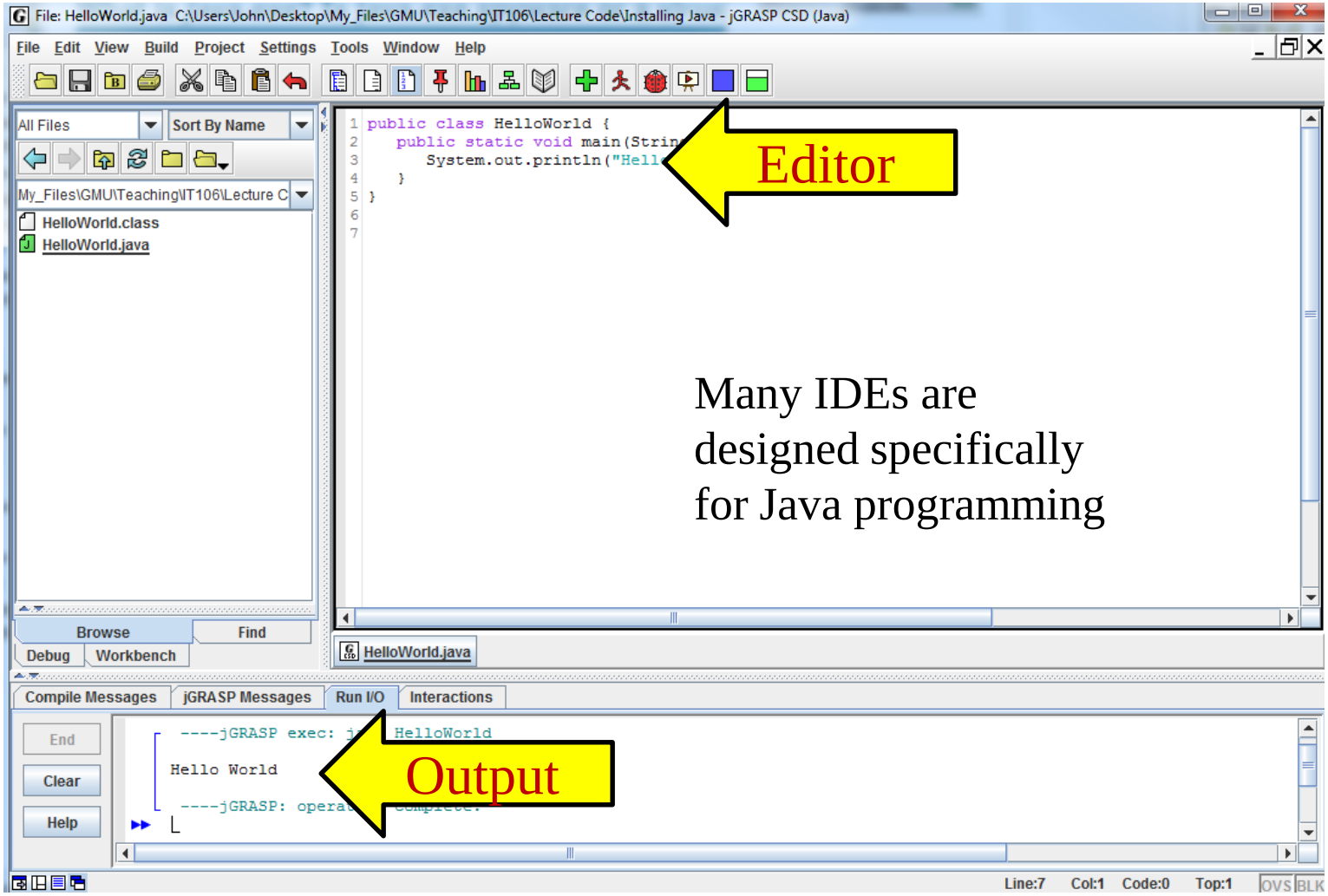
Install JDK

- **To be able to create Java programs, you must install the Java Development Kit (JDK)**
- **Download: http://www.oracle.com/java/**
- **Common location after installation will be:**
  - C:\Program Files\Java\jdk_____ (a set of numbers)
  - The set of numbers will vary with the release
- **The JDK includes programs such as:**
  - javac.exe (Java compiler)
  - javadoc.exe (Javadoc generator)
  - java.exe (executes Java applications)

Java Integrated Development Environment (IDE)

- **There are many free programming tools available for Java**
- **jGrasp (http://www.jgrasp.org/) is strongly recommended for this course instead of anything else (e.g. NetBeans, Eclipse, etc.)**
- **Source code editor helps programming by:**
  - Lists line numbers
  - Color codes special words
  - Helps with indenting for readability
- **Output window**

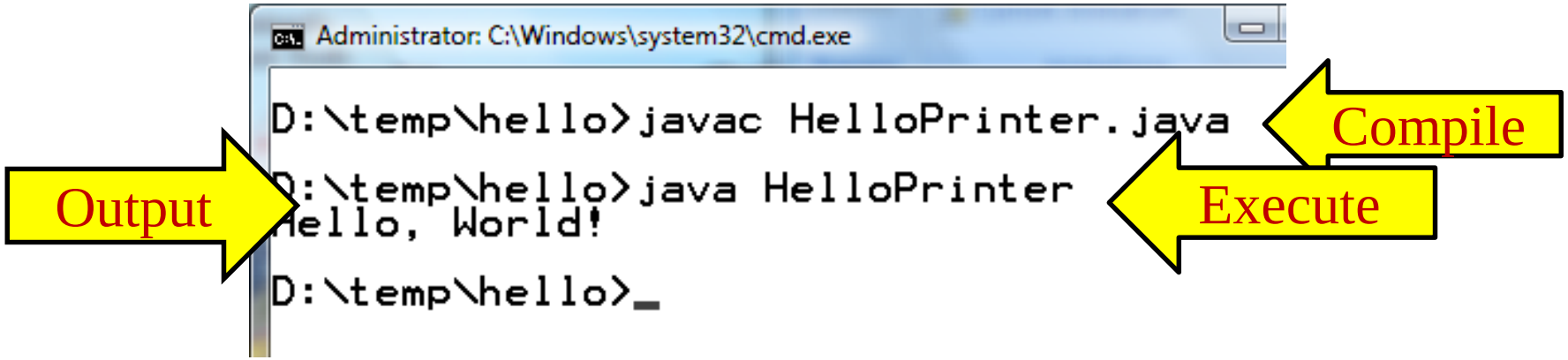
jGrasp: An Example IDE



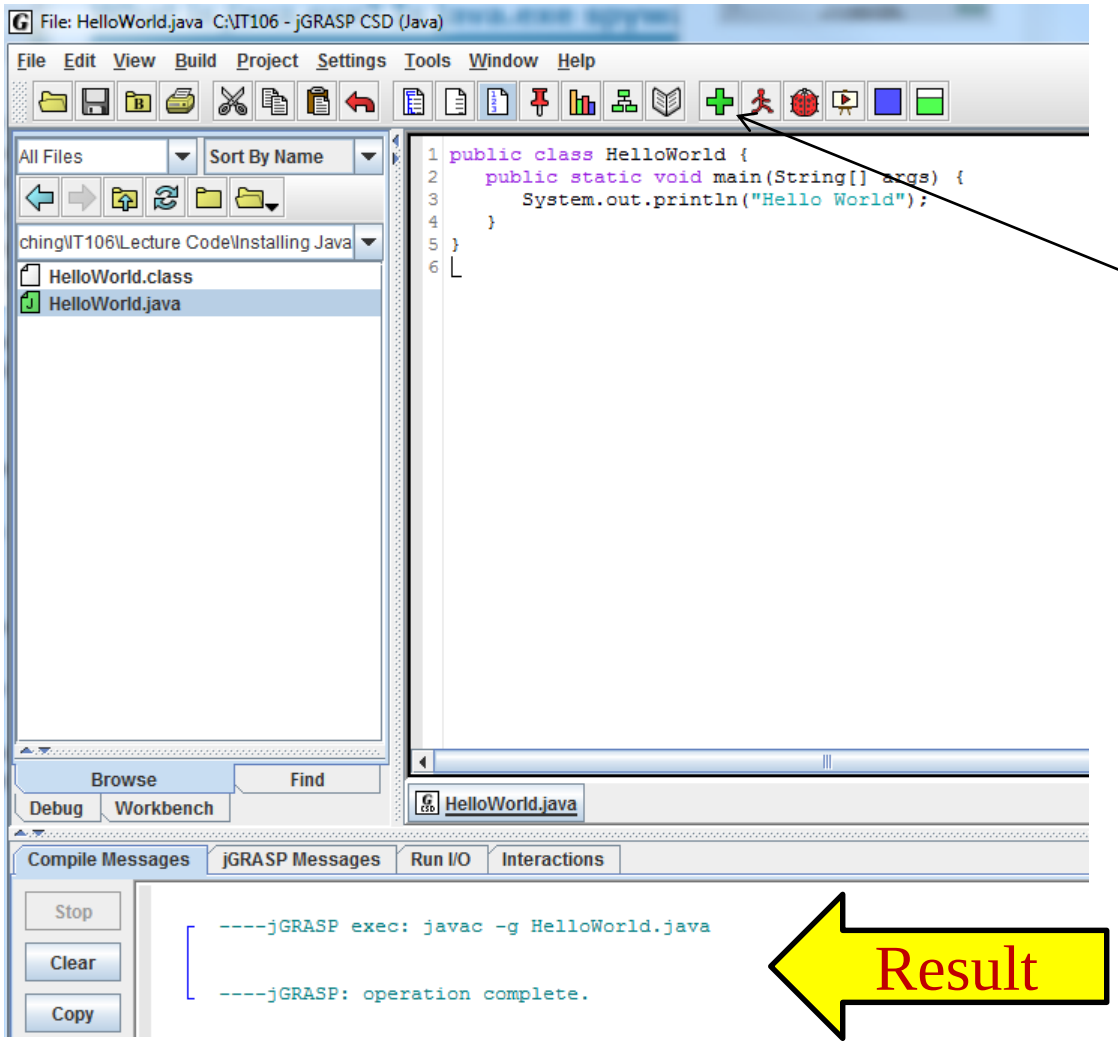
Many IDEs are designed specifically for Java programming

Text Editor Programming

- **You do not need to have an IDE**
- **You can use a simple text editor, such as Notepad to write your source code**
- **Assuming a code file named, *HelloPrinter.java*, you can use a command prompt to:**
  - Compile the program
  - Execute the program

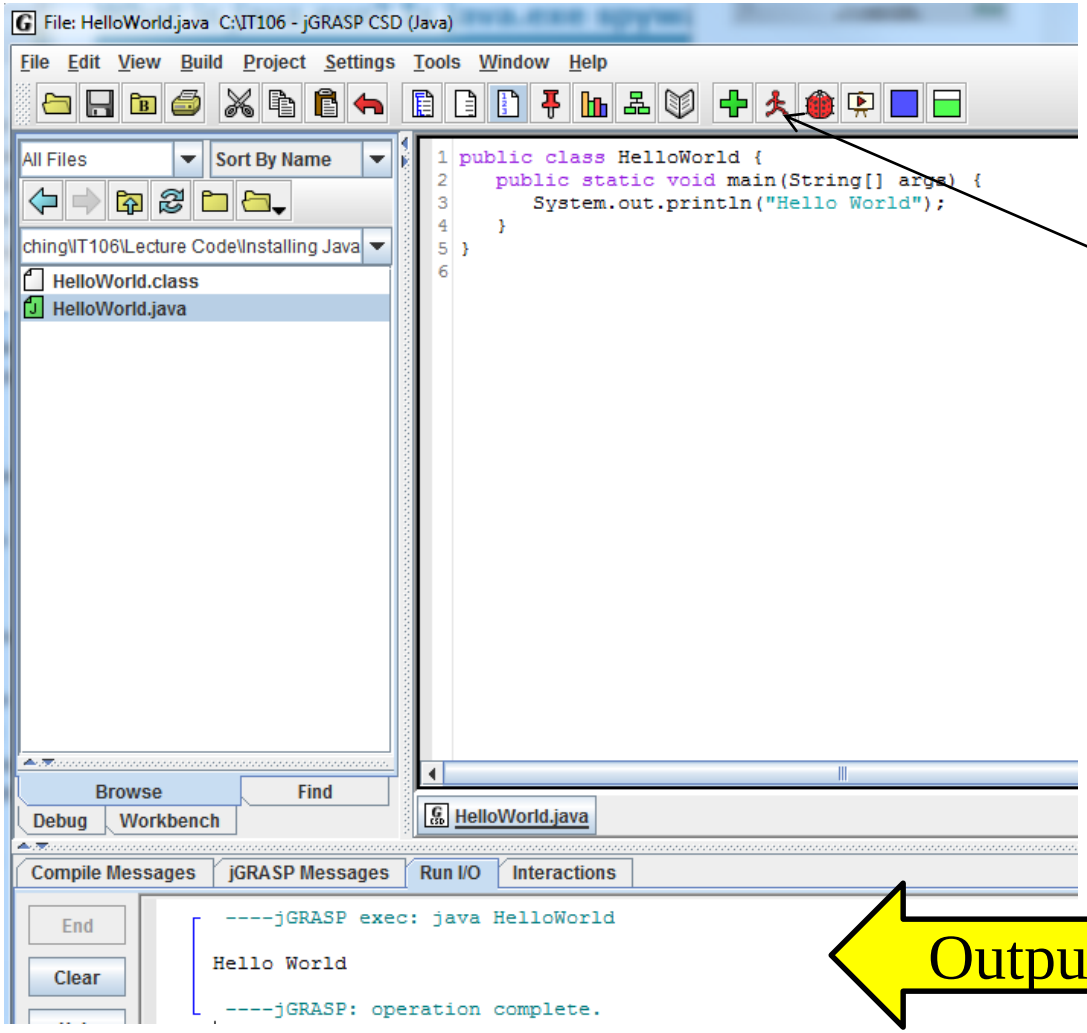


jGrasp: Compiling a Program



Click the "+" icon to compile the program

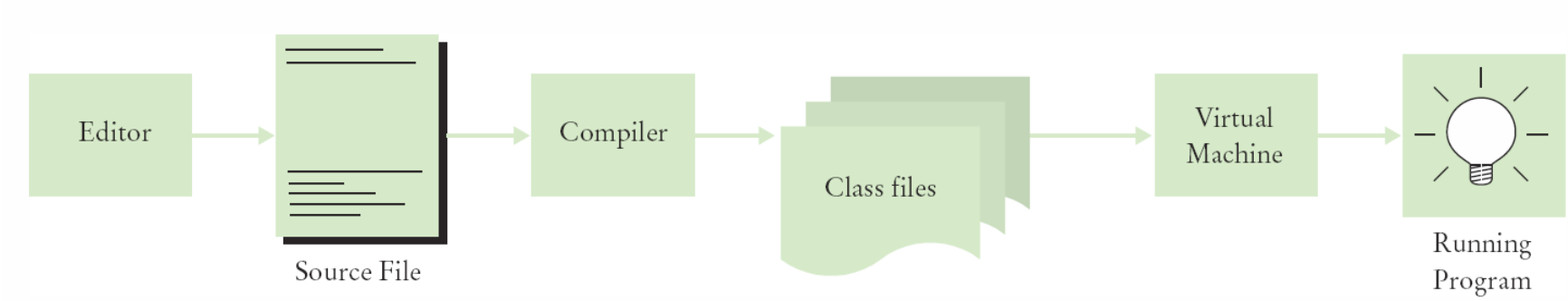
jGrasp: Running a Program



Output

Click the red "Running Man" icon to execute the program

Source Code to Running Program



- The compiler generates .class files for each .java file, which contains machine-readable instructions for the Java Virtual Machine (JVM)
- .class files contain 'byte code' that are machine readable and uneditable

Organize Your Work

- **Source code is stored in .java files**
- **Create one folder per program**
  - A program can have many .java files
- **Be sure you know where the IDE stores your files**
- **"**IT Happens"**
  - Backup your work!

Backup your work to a Flash Drive, external hard drive, network drive, or cloud storage that is backed up nightly.

Anatomy of a Java Program

- Comments
- Package
- Reserved words
- Modifiers
- Statements
- Blocks
- Classes
- Methods
- The main method

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 */.

Package

The second line in the program (package chapter1;) specifies a package name, chapter1, for the class Welcome. IDE compiles the source code in Welcome.java, generates Welcome.class, and stores Welcome.class in the chapter1 folder.

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. Other reserved words in Listing 1.1 are public, static, and void. Their use will be introduced later in the book.

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. Modifiers are discussed in Chapter 6, “Objects and 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. The mystery of the class will continue to be unveiled throughout this course. For now, though, understand that a program is defined by using one or more classes.

Methods

What is System.out.println? It is a method: a collection of statements that performs a sequence of operations to display a message on the console. It can be used even without fully understanding the details of how it works. It is used by invoking a statement with a string argument. The string argument is enclosed within parentheses. In this case, the argument is "Welcome to Java!" You can call the same println method with a different argument to print a different message.

main Method

The main method provides the control of program flow. The Java interpreter executes the application by invoking the main method.

The main method looks like this:

```
public static void main(String[] args) {
  // Statements;
}
```

Identifiers

- An identifier is a sequence of characters that consist of letters, digits, underscores (_), and dollar signs ($).
- An identifier must start with a letter, an underscore (_), or a dollar sign ($). It cannot start with a digit.
  - An identifier cannot be a reserved word. (See Appendix A, "Java Keywords," for a list of reserved words).
- An identifier cannot be `true`, `false`, or `null`.
- An identifier can be of any length.

Variables

```
// Compute the first area
radius = 1.0;
area = radius * radius * 3.14159;
System.out.println("The area is " +
  area + " for radius "+radius);

// Compute the second area
radius = 2.0;
area = radius * radius * 3.14159;
System.out.println("The area is " +
  area + " for radius "+radius);
```

Constants

```
final datatype CONSTANTNAME = VALUE;

final double PI = 3.14159;
final int SIZE = 3;
```

Questions?

Content

- Components of the Java Environment
- Your First Java Program
- Variables and Primitive Data Types
- Selection Statements
- Loop Statements
- Methods

Hello World: Your First Java Program

- **Below is a traditional "Hello World" program in Java**
  - The name of the file is HelloWorld.java
- **Typing the program into your IDE would be good practice!**
  - Be careful of spelling
  - JaVa iS CaSe SeNsItiVe
  - Java uses special characters, such as curly braces {} and parentheses ()
  - Java ignores whitespace

```
1 public class HelloWorld {
2     public static void main(String[] args) {
3         System.out.println("Hello World");
4     }
5 }
```

Analyzing Your First Java Program

```
1 public class HelloWorld {
2     public static void main(String[] args) {
3         System.out.println("Hello World");
4     }
5 }
```

- **Line 1: Declares a class called HelloWorld**
  - Java programs are constructed with one or more classes
- **Line 2: Declares a method called main**
  - Every Java program has exactly one **main** method
  - The **main** method is the entry point where the program starts
- **Line 3: Method System.out.println outputs "Hello World"**
  - System.out is part of the Java API
  - All statements must end with a semicolon

Calling Java API Methods

- **Note the Line:**

```
3        System.out.println("Hello World");
```

- **It shows how to "call" a method from the Java API (System.out.println)**
  - Code that somebody else wrote for you
  - Notice the dots (periods)
  - Parentheses surround the arguments that you "pass" to a method
    - We are passing a String "Hello World"
    - Using double quotes denotes a string
  - You can also print numeric values
    - Example: System.out.println(106);
    - Example: System.out.println(3 + 4);
    - Note that numbers are not quoted

More on the println Method

- **The println method prints a string or a number and then starts a new line**
  - System.out.println("Hello");
  - System.out.println("World!");

```
Hello
World!
```

- **A similar function that does not print a new line is print**
  - System.out.print("00");
  - System.out.println(3+4);

```
007
```

Errors

- **Syntax Errors**
  - Examples
    - Misspelling, capitalization, punctuation
    - Ordering of statements, matching of braces/parentheses
  - No .class file is generated by the compiler
  - Correct the first error listed, then compile again
- **Logic Errors**
  - Program runs, but produces unintended results
  - Check your algorithm for the logic you have included
- **Runtime Errors**
  - Causes the program to crash immediately, such as divide by zero
  - Check your algorithm to make sure you have handled all cases

Syntax Errors

```
1 public class HelloWorld {
2     public static void main(String[] args) {
3         System.out.println("Hello World");
4     }
5 }
```

- **What happens if you:**
  - misspell a word — System.**ou**.println
  - don't capitalize a word — **s**ystem.out.println
  - leave out a word — System.println
  - forget a semicolon — Remove ; at the end of line 3
  - don't match a curly brace — Remove line 5
- **Try each of these to see what happens when you try to compile to get practice in dealing with compiler error messages**

Logic Errors

```
1 public class HelloWorld {
2     public static void main(String[] args) {
3         System.out.println("Hello World");
4     }
5 }
```

- **What happens if you:**
  - misspell the output ("Hello Word")
  - forget to output Remove line 3
- **In these cases, the program will compile and run**
  - The output may not be as expected

Runtime Errors

```
1 public class HelloWorld {
2     public static void main(String[] args) {
3         System.out.println(1/0);
4     }
5 }
```

- **What happens if you:**
  - Divide by zero System.out.println(1/0)
- **In these cases, the program will compile, but when it runs, it will crash**

```
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at HelloWorld.main(HelloWorld.java:3)

----jGRASP wedge2: exit code for process is 1.
----jGRASP: operation complete.
```

Questions?

Content

- Components of the Java Environment
- Your First Java Program
- Variables and Primitive Data Types
- Selection Statements
- Loop Statements
- Methods

VARIABLES

Declaring Variables

- **Most computer programs hold temporary values in named storage locations (variables)**
  - They are named for easy access
- **There are many different types and sizes of storage available**
  - Each is used depending on what you are trying to store
- **Variables are declared by telling the compiler**
  - What type (and size, if applicable) of variable you need
  - What name you will use to refer to it

Variable Example: Soda Deal

- **Problem**
  - Soft drinks are sold in cans and bottles. A store offers a six-pack of 12-ounce cans for the same price as a two-liter bottle. Which should you buy? (Note: 12 fluid oz. equals approximately 0.355 liters)

Variable	Data Type	Value
Number of cans per pack	Whole number	6
Ounces per can	Whole number	12
Ounces per bottle	Number with fraction	67.606 (12 oz. * 2 liters / 0.355 liters)

Variable Contents

- **Each variable has an identifier (name) and contents**
- **You can (optionally) set the contents of a variable when you declare it**
  - Example: int cansPerPack = 6;

cansPerPack

6

- **Imagine a parking space in a parking garage**
  - Identifier (Name): J053
  - Contents: Bob's BMW

A variable is a storage location with a name

Syntax: Variable Declaration

- **When declaring a variable, you often specify an initial value**
- **This is where you tell the compiler the type (and size if appropriate) of data it will hold**

See page 35 for rules and examples of valid names.

Types introduced in this chapter are the number types `int` and `double` (page 34) and the `String` type (page 60).

```
int cansPerPack = 6;
```

A variable declaration ends with a semicolon.

Use a descriptive variable name. See page 38.

Supplying an initial value is optional, but it is usually a good idea. See page 37.

Example Variable Declarations

Table 1 Variable Declarations in Java

Variable Name	Comment
`int cans = 6;`	Declares an integer variable and initializes it with 6.
`int total = cans + bottles;`	The initial value need not be a constant. (Of course, `cans` and `bottles` must have been previously declared.)
`bottles = 1;`	**Error:** The type is missing. This statement is not a declaration but an assignment of a new value to an existing variable—see Section 2.2.
`int bottles = "10";`	**Error:** You cannot initialize a number with a string.
`int bottles;`	Declares an integer variable without initializing it. This can be a cause for errors—see Common Error 2.1 on page 37.

Naming Variables

- **The name should clearly describe the purpose**
  - In this case it is better to be verbose than not
  - Example: canVolume is better than cv
- **Simple Rules to Choosing a Name:**
  - Use only letters, numbers, or the underscore (_)
    - Don't start with a number
  - Separate words with "camelCase" notation
    - Example: canVolume, cansPerPack
  - Variable names are case-sensitive
  - Don't use Java reserved words
    - Example: can't have a variable named int
  - Variables start with lower case / Class names start with upper case
    - Not a requirement, but a general best practice

Naming Variables (Cont'd)

- **Legal and illegal variable names**

Table 3 Variable Names in Java

Variable Name	Comment
`canVolume1`	Variable names consist of letters, numbers, and the underscore character.
`x`	In mathematics, you use short variable names such as *x* or *y*. This is legal in Java, but not very common, because it can make programs harder to understand (see Programming Tip 2.1 on page 38).
⚠ `CanVolume`	**Caution:** Variable names are case sensitive. This variable name is different from `canVolume`, and it violates the convention that variable names should start with a lowercase letter.
⊘ `6pack`	**Error:** Variable names cannot start with a number.
⊘ `can volume`	**Error:** Variable names cannot contain spaces.
⊘ `double`	**Error:** You cannot use a reserved word as a variable name.
⊘ `ltr/fl.oz`	**Error:** You cannot use symbols such as / or .

Variable Data Types

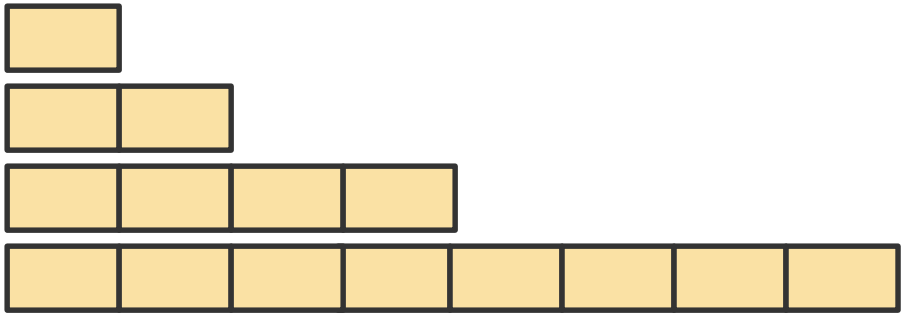
- **Data types are used to tell the compiler what type of data you are trying to store**
  - Helps to allocate the correct amount of memory
- **Common Types**
  - A whole number (no fraction) – **int**
  - A number with a fraction part – **double**
  - A single character – **char**
  - A word (a group of characters) – **String**
- **The type is specified when declaring the variable**
  - Example: **int** cansPerPack = 6;
  - Example: **double** canVolume = 12.0;
  - Example: **String** name = "Jack";
- **Back to the garage analogy, parking spaces may be different sizes for different types of vehicles**
  - E.g. bicycle, motorcycle, full size, van, etc.

Variable Data Types (Cont'd)

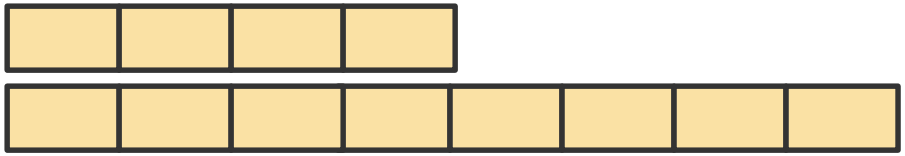
- **Integer Types (Whole numbers, no fractions)**
  - `byte`: a very small number (-127 to +128)
  - `short`: a small number (-32,768 to +32,767)
  - `int`: a large number (-2,147,483,648 to +2,147,483,647)
  - `long`: a huge number
- **Floating Point Types**
  - `float`: a huge number with decimal places
  - `double`: a more precise, floating type, used for calculations
- **Other Types**
  - `boolean`: true or false
  - `char`: one symbol in single quotes, such as: 'a'

Variable Data Type Storage

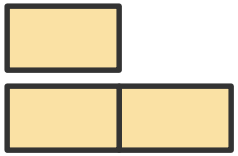
- **Integer Types (Whole numbers, no fractions)**
  - `byte`:
  - `short`:
  - `int`:
  - `long`:



- **Floating Point Types**
  - `float`:
  - `double`:



- **Other Types**
  - `boolean`:
  - `char`:



Number Literal Examples in Java

Use the `double` type for floating-point numbers.

Table 2 Number Literals in Java

Number	Type	Comment
6	`int`	An integer has no fractional part.
-6	`int`	Integers can be negative.
0	`int`	Zero is an integer.
0.5	`double`	A number with a fractional part has type `double`.
1.0	`double`	An integer with a fractional part .0 has type `double`.
1E6	`double`	A number in exponential notation: 1×10^6 or 1000000. Numbers in exponential notation always have type `double`.
2.96E-2	`double`	Negative exponent: $2.96 \times 10^{-2} = 2.96 / 100 = 0.0296$
100,000		**Error:** Do not use a comma as a decimal separator.
3 1/2		**Error:** Do not use fractions; use decimal notation: 3.5

Java Comments

- **There are three forms of comments**
  - // a single line comment (or rest of line to the right)
  - /*
    multi-line comment – all comments within /* */
    */
  - /**
    multi-line Javadoc comments – used to automatically generate documentation
    */
- **The compiler ignores commented code**
- **Use comments at the beginning of each program and within the program to clarify details of the code**

Java Comment Example

```
1  /**
2     The purpose of this program is to compute the volume (in liters) of a six-pack
3     of soda cans. There is no user input, but the program will output the number
4     of liters in a six-pack of 12-ounce cans.
5  */
6
7  public class SodaCanVolume {
8     public static void main(String[] args) {
9        int cansPerPack = 6;
10       double canVolume = 0.355; // Number of liters in a 12-ounce can
11
12       // Print number of liters in a 12-ounce can
13       System.out.println("A six-pack of 12-ounce cans contains: ");
14       System.out.print(cansPerPack * canVolume);
15       System.out.println(" liters");
16    }
17 }
```

External Documentation

Internal Documentation

- **Lines 1-5 are Javadoc comments for the class "SodaCanVolume"**
- **Line 10 uses a single-line comment to clarify the unit of measurement**
- **Line 12 shows what the following three lines after it are doing**

Common Variable Errors

- **Undeclared Variables**
  - You must declare a variable before you use it: (i.e. above in the code)

```
double canVolume = 12 * literPerOunce; // ??
double literPerOunce = 0.0296;
```

- **Uninitialized Variables**
  - You must initialize (i.e. set) a variable's contents before you use it

```
int bottles;
int bottleVolume = bottles * 2;   // ??
```

Common Variable Errors (Cont'd)

- **Overflow**
  - The storage for the variable cannot correctly hold the value
  - Example: Remember the int data type can store values in the range of -2,147,483,648 to +2,147,483,647
    - `int oneThousand = 1000;  // OK`
    - `int oneMillion = 1000 * oneThousand;  // OK`
    - `int oneBillion = 1000 * oneMillion;  // OK`
    - `System.out.println(3 * oneBillion);  // ??`
  - Output will print: -1294976296
  - Why?
    - The result (3 billion) overflowed the capacity of an int, truncated the value, and provided something useless
  - To fix this, use a long (if integer) or a double (if floating point)

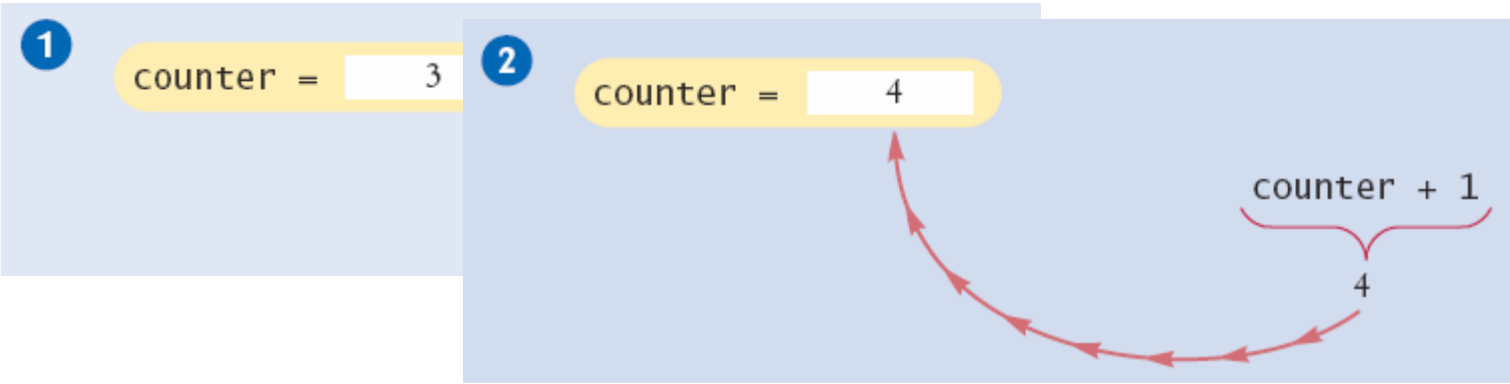
Variable Assignment

- **You can use the assignment operator (=) to place a new value into a variable**

```
int cansPerPack = 6; //declare and initialize
cansPerPack = 8;      //assignment
```

- **Warning! The = sign is NOT used for comparison**
  - It copies the value on the right side into the variable on the left side
    - The variable <u>MUST</u> be on the left side for assignment
  - Comparisons will be covered a bit later

Incrementing a Variable



- **Code:** `counter = counter + 1;`
- **Steps:**
  1. Do the right hand of the assignment first
     - Find the value stored in counter, then add 1 to it
  2. Store the result in the variable named on the left side of the assignment operator (counter in this case)

Shorthand for Incrementing

- **Incrementing (+1) and decrementing (-1) integer types is so common that there is a shorthand version for each**

Long Way	Shortcut
`counter = counter + 1;`	`counter++;`
`counter = counter - 1;`	`counter--;`

Modifying a Variable

- **Assumes counter has already been declared using `int counter;`**

Table 5 Modifying a Variable

Statements	Contents of counter	Comments
`counter = 1;`	1	The previous content of the variable has been replaced.
`counter = counter + 1;`	2	Adds 1 to `counter`. Note that `=` is not mathematical equality.
`counter++;`	3	`++` is a shorthand for adding 1 to a variable.
`counter--;`	2	`--` is a shorthand for subtracting 1.
`int counter = 4;`		**Error:** This is not an assignment but an attempt to declare a second variable named `counter`.

Syntax: Assignment

- **Review of the assignment statement**
  - The value on the right is copied to the variable on the left

This is an initialization of a new variable, NOT an assignment.

```
double total = 0;
   .
   .
total = bottleVolume;
   .
   .
total = total + canVolume;
```

This is an assignment.

The name of a previously defined variable

The expression that replaces the previous value

The same name can occur on both sides. See Figure 1.

Constants

- **It is a good practice to declare values that will not change during program execution as constants**
- **Use the reserved word `final` before the type in the declaration**
  - Example: `final double BOTTLE_VOLUME = 1.75;`
  - They can then be used like any other variable

```
double volume = bottles * BOTTLE_VOLUME;
```

- **Constants are usually declared near the beginning of the program or class**

You cannot assign a new value to a constant at run-time.

INPUT/OUTPUT

Using a Graphical User Interface

- **Previously for output, we have been using System.out.println to print to the IDE or a console window**
- **Users are now used to seeing graphical interfaces instead of just a text-based command prompt**
- **For now, don't worry about the details**
  - Pay attention to the format of the code

Using a GUI for Output

- **JOptionPane: Package that provides dialog boxes for GUI output**
  - Benefit: Makes your program look "prettier"
- **Implementation**
  - Import Java package containing JOptionPane class
    - Use: `import javax.swing.JOptionPane;` at the top of your .java file
  - Use showMessageDialog() method, which is similar to println(), but takes in two pieces of data instead
    - Pass null and the String you want to output

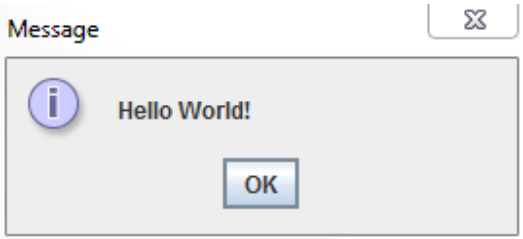
Using a GUI for Output

- **Old**

```java
public class HelloPrinter {
   public static void main(String[] args) {
      System.out.println("Hello World!");
   }
}
```

- **New**

```java
import javax.swing.JOptionPane;

public class HelloPrinterGUI {
   public static void main(String[] args) {
      JOptionPane.showMessageDialog(null, "Hello World!");
   }
}
```



Reading Input

- **You frequently will need to ask the user for input (i.e. prompt them) and then save what was entered**
  - We will be reading input from the keyboard

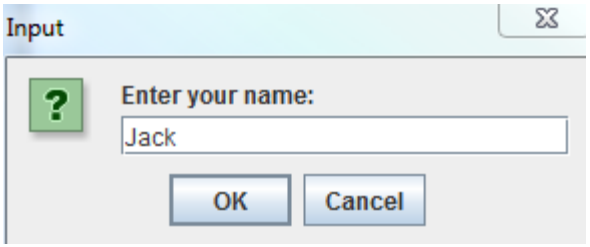
Reading Input (Cont'd)

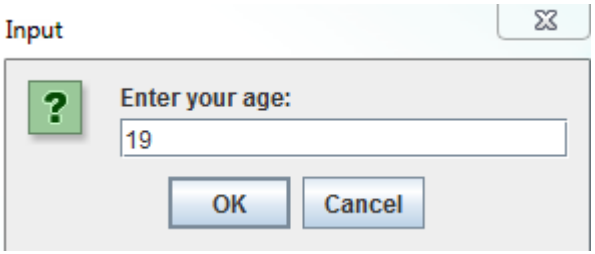
- **Three step process to do this, using a GUI**
  - Import Java package containing JOptionPane class
    - Use: `import javax.swing.JOptionPane;` at the top of your .java file
  - Use methods of the JOptionPane class to show an input dialog
    - Use: `JOptionPane.showInputDialog("MessageToUser");`
  - In the same line, convert the input entered into the correct data type if the data type is numeric and calculations will be needed

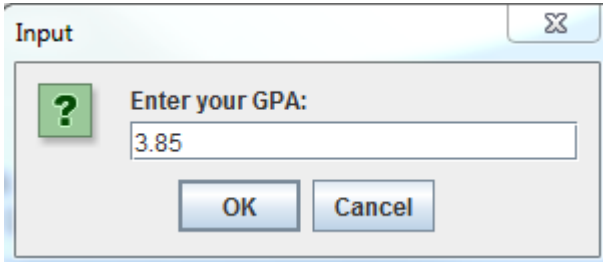
```
String name = JOptionPane.showInputDialog("Enter your name:");
int age = Integer.parseInt(JOptionPane.showInputDialog("Enter your age:"));
double gpa = Double.parseDouble(JOptionPane.showInputDialog("Enter your GPA:"));
```

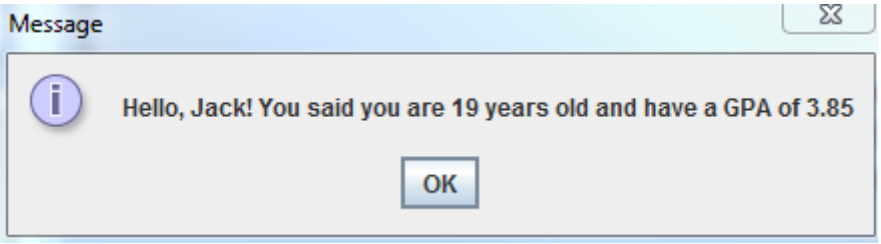
Example: Reading Input

```java
import javax.swing.JOptionPane;

public class GUIInputExample {
   public static void main(String[] args) {
      String name = JOptionPane.showInputDialog("Enter your name:");
      int age = Integer.parseInt(JOptionPane.showInputDialog("Enter your age:"));
      double gpa = Double.parseDouble(JOptionPane.showInputDialog("Enter your GPA:"));

      JOptionPane.showMessageDialog(null, "Hello, " + name + "! You said you are "
      + age + " years old and have a GPA of " + gpa);
   }
}
```



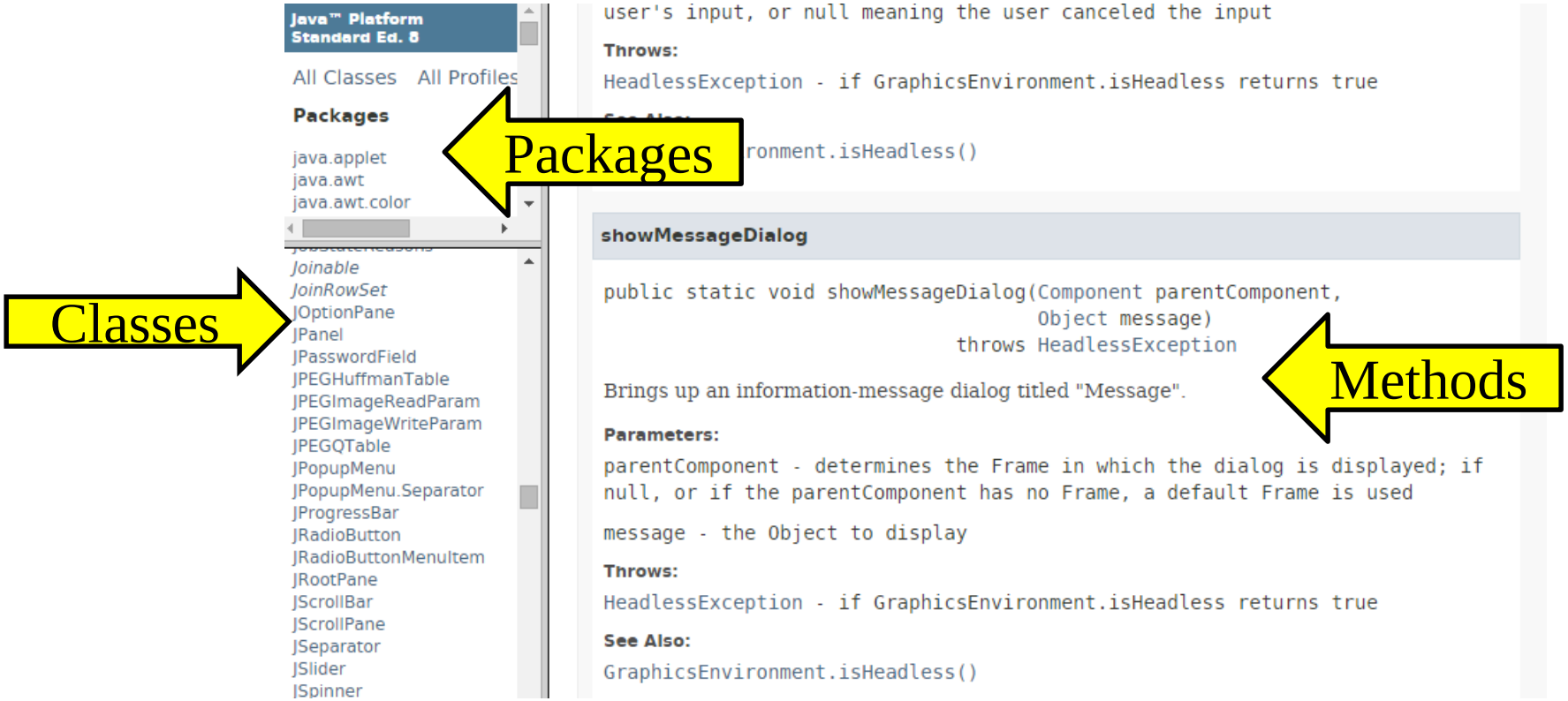






Tip: Java API Documentation

- **Lists all of the classes and methods within the Java API**
  - On the Web: http://docs.oracle.com/javase/8/docs/api/



ARITHMETIC OPERATIONS

Arithmetic Operations

- **Java supports all of the same basic calculator operations**
  - Addition, subtraction, multiplication, division
- **However, Java expressions may only be written on one line**
- **Precedence is similar to Algebra**
  - Remember PEMDAS
    - Parentheses, exponents, multiplication/division, addition/subtraction

No	Yes
$\frac{a+b}{2}$	`(a + b) / 2`

Integer Division and Remainders

- **If both operands are integer types, you need to be careful not to lose precision**

```
int first = 7, second = 4, answer;
answer =  first / second;  // answer is 1!
```

  – The result is an integer. The fraction was lost.

- **To find the fractional part, use the modulo operator (%)**

```
int first = 7, second = 4, answer, remainder;
answer =  first / second;
remainder =  first % second;  // set to 3
```

- **You could also use floating-point data types, instead**

Powers and Roots

- **There are no symbols for powers and roots**
  - But, methods exist in the Java Math class

$$b \times \left(1 + \frac{r}{100}\right)^n$$

Java

```
b * Math.pow(1 + r / 100, n)
```

Table 6 Other Mathematical Methods and Constants

Method	Description
`Math.sin(x)`	sine of x (x in radians)
`Math.cos(x)`	cosine of x
`Math.tan(x)`	tangent of x
`Math.log10(x)`	(decimal log) $\log_{10}(x)$, $x > 0$
`Math.abs(x)`	absolute value $\lvert x \rvert$

Powers and Roots (Cont'd)

Mathematical Expression	Java Expression	Comments
$\frac{x+y}{2}$	`(x + y) / 2`	The parentheses are required; `x + y / 2` computes $x + \frac{y}{2}$.
$\frac{xy}{2}$	`x * y / 2`	Parentheses are not required; operators with the same precedence are evaluated left to right.
$\left(1 + \frac{r}{100}\right)^n$	`Math.pow(1 + r / 100, n)`	Use `Math.pow(x, n)` to compute x^n.
$\sqrt{a^2 + b^2}$	`Math.sqrt(a * a + b * b)`	`a * a` is simpler than `Math.pow(a, 2)`.
$\frac{i+j+k}{3}$	`(i + j + k) / 3.0`	If i, j, and k are integers, using a denominator of 3.0 forces floating-point division.
π	`Math.PI`	`Math.PI` is a constant declared in the `Math` class.

CONVERTING AND FORMATTING DATA

Converting Between Data Types

- **It is safe to convert a value from an integer data type to a floating point data type**
  - No decimal points (precision) is lost
- **Going the other way can be dangerous**
  - All fractional information is lost
  - To "force" a conversion, variables can be **cast** from one type to another
  - Example

```
double balance = total + tax;
int dollars = (int) balance;
```

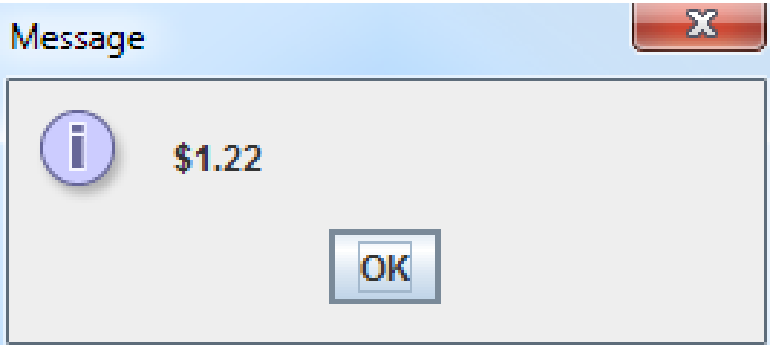
    - The fractional part is discarded (not rounded)
    - If you do not use the cast, the compiler will generate a syntax error

Formatted Output

- **Outputting floating point values can look strange, such as at gas stations**
  - Example: Price per liter = 1.21997
- **To control the output appearance we can create and format a String containing a number through a format specifier**
- **Once the string has been formatted, we can output it**
- **More detailed ways you can format your output**
  - *Note:* Not all format specifiers are supported using JOptionPane

Formatted Output (Cont'd)

```java
import javax.swing.JOptionPane;

public class FormattedOutput {
   public static void main(String[] args) {
      double pricePerLiter = 1.21997;

      // Setup a new variable of type: String
      // and format the value to two decimal places which can be used in output
      JOptionPane.showMessageDialog(null, String.format("$%.2f", pricePerLiter));
   }
}
```



Common Errors

- **Unintended Integer Division**

```java
import javax.swing.JOptionPane;

public class UnintendedIntegerDivision {
   public static void main(String[] args) {
      final int NUM_GRADES = 3;
      int grade1 = Integer.parseInt(JOptionPane.showInputDialog("Enter grade 1:"));
      int grade2 = Integer.parseInt(JOptionPane.showInputDialog("Enter grade 2:"));
      int grade3 = Integer.parseInt(JOptionPane.showInputDialog("Enter grade 3:"));

      double average = (grade1 + grade2 + grade3) / NUM_GRADES;

      JOptionPane.showMessageDialog(null, "Grade: " + average);
   }
}
```

- **Why an error?**
  - Since all of the data types are integers, the compiler performs integer division
  - Then the resulting integer is assigned to a double
  - When decimal points are important, use a floating-point type

Common Errors (Cont'd)

- **Unbalanced Parentheses**
- **The count of (and) must match**
  - Incorrect: `-(b * b - (4 * a * c) ) ) / 2 * a)`
- **It can be difficult to keep track**
  - Trick:
    - Count (as + 1 and) as -1. Goal : 0
  - Example:

```
-(b * b - (4 * a * c) ) ) / 2 * a)
 1        2         1 0 -1      -2
```

STRINGS

Strings

- The String data type stores a set of characters
- Once you have a string variable you can use methods on it, such as finding the length

```
1  import javax.swing.JOptionPane;
2
3  public class FindStringLength {
4     public static void main(String[] args) {
5        String firstName = "Mary";
6        int nameLength = firstName.length();
7
8        JOptionPane.showMessageDialog(null, nameLength);
9     }
10 }
```

- The maximum length of a string is the size of an integer
- "" is considered an empty String (length 0)

String Concatenation (+)

- You can add or concatenate one String onto the end of another

```
1 public class StringConcatenation1 {
2    public static void main(String[] args) {
3       String firstName = "James";
4       String lastName = "Bond";
5       String name = firstName + lastName; // JamesBond
6       String name2 = firstName + " " + lastName; // James Bond
7    }
8 }
9
```

- You can also concatenate a number to a

```
1 public class StringConcatenation2 {
2    public static void main(String[] args) {
3       String firstName = "Agent";
4       int agentNumber = 7;
5       String agent = firstName + agentNumber; // Agent7
6    }
7 }
```

String Escape Sequences

- **How would you store a " within a String?**
  - Preface the " with a \ inside the double quoted String
- **How would you print a backslash?**
  - Preface the \ with another \
- **Special characters inside String**
  - To create a new line, use \n

```
1 public class StringConcatenation3 {
2    public static void main(String[] args) {
3       String doubleQuote = "You are \"special\"";
4       String domainUser = "DOMAIN\\username";
5       String helloWorld = "Hello\nWorld!";
6    }
7 }
8
```

Practice Problem

- **Scenario**
  - For the course, the Syllabus states the final grade is calculated as an average of three, equally weighted exams. A program is required to prompt the instructor to enter the grades for each of the exams, which will then calculate and display to the screen the final numeric grade
- **To Do:**
  - Create a defining diagram
  - Create a solution algorithm using pseudocode
  - Writing the program with Java

Practice Problem (cont'd)

- **Create a defining diagram**

Input	Processing	Output
number1 number2 number3	• Prompt for number1, number2, number 3 • Read number1, number2, number3 • Add three numbers together • Print total of three numbers	total

- **Create a solution algorithm using pseudocode**

```
BEGIN ADD_THREE_NUMBERS
1         PROMPT for number1
2         READ number1
3         PROMPT for number2
4         READ number2
5         PROMPT for number3
6         READ number3
7         SET total = number1 + number2 + number3
8         PRINT 'The total is ', total
END
```

Questions?