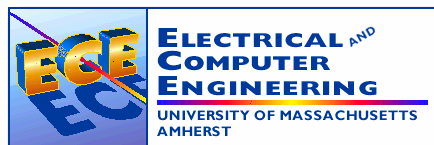

ECE 122

Engineering Problem Solving with Java

Lecture 2

Program Development



Outline

- **Problem: How do I define and run a Java program?**
 - What does the program do with my text?
 - What makes Java different?

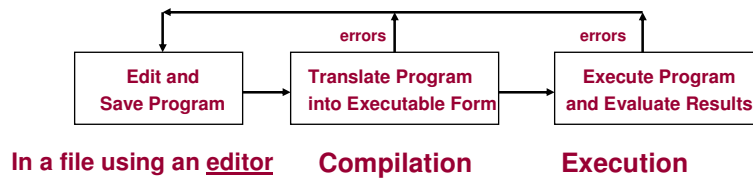
- **Representing computation with objects**

- **Representing data**
 - Characters
 - Integers
 - Floating point

- **Making assignments to variables**

Program Development

- The mechanics of developing a program include several activities
 - Writing the program in a specific programming language (such as Java)
 - Translating the program into a form that the computer can execute
 - Investigating and fixing various types of errors that can occur
- Software tools can be used to help with all parts of this process



Translation of Source Code

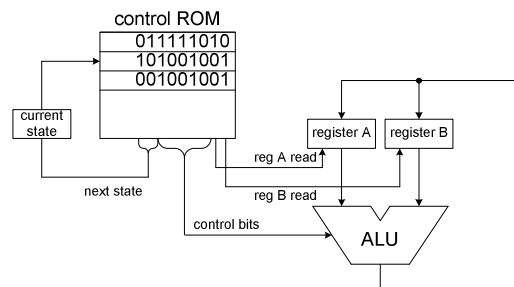
- The translation of source code into machine language
- Subsequent execution on a particular type of CPU can occur in a variety of ways.
 - By using a **compiler**.
 - By using an **interpreter**.
 - By using both a **compiler and interpreter**
 - This is how Java does it.

A Compiler

- A compiler is a program that translates code from one language to an equivalent code in another language.
- The original code is called **source-code**.
 - The language into which it is translated is called the **target language**.
 - For many traditional compilers, the source code is translated directly into a particular machine language.
 - The translation process occurs once and the resulting executable program can be run whenever needed.

Java Programming Language

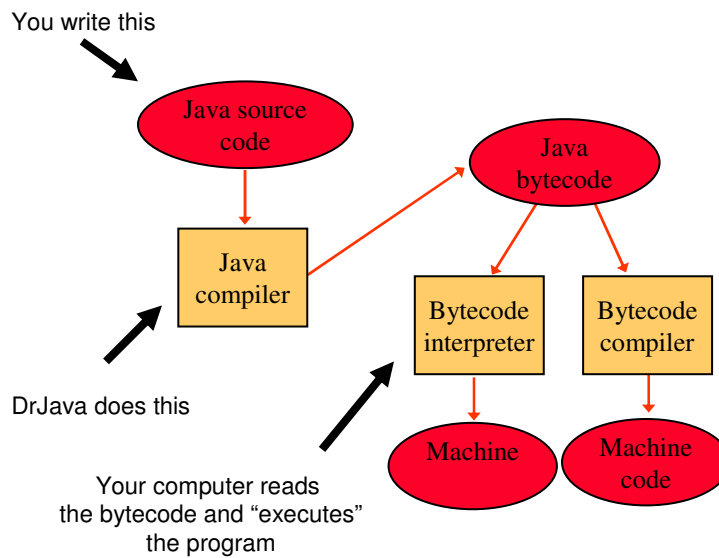
- Each type of CPU executes only a particular *machine language*
- A program must be translated into machine language before it can be executed
- A *compiler* is a software tool which translates *source code* into a specific target language
- Often, that target language is the machine language for a particular CPU type



An Interpreter

- **An interpreter** is similar to a compiler but has important differences...
 - **An interpreter performs the translation and execution activities.**
 - A small part of the source code, such as one statement, is translated and executed.
 - Then another part is translated and executed, and so on.
 - **The program runs more slowly because the translation process occurs during each execution.**
 - Each statement is translated, then executed immediately

Java Translation



Java Translation

- The Java compiler translates Java source code into a special representation called *bytecode*
- Java bytecode is not the machine language for any traditional CPU
- Another software tool, called an *interpreter*, translates bytecode into machine language and executes it
- Therefore the Java compiler is not tied to any particular machine
- Java is considered to be *neutral*

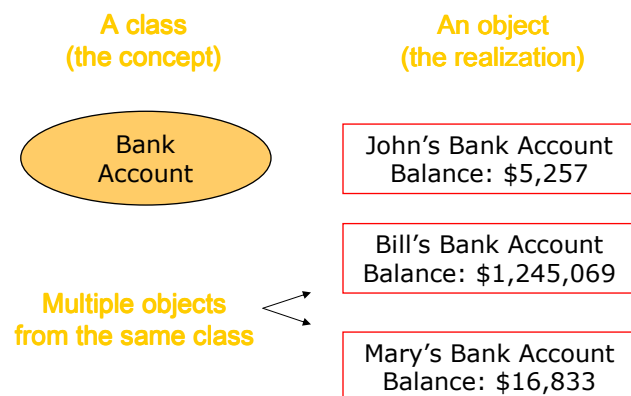
Java Program Structure

- A program is made up of one or more *classes*
- A class contains one or more *methods*
- A method contains program *statements*
- A Java application always executes the *main* method

Classes

- An object is defined by a *class*
- A class is the blueprint of an object
- The class uses methods to define the behaviors of the object
- The class that contains the main method of a Java program represents the entire program
- A class represents a concept, and an object represents the embodiment of that concept
- Multiple objects can be created from the same class

Objects and Classes



Character Strings

- A string of characters can be represented as a *string literal* by putting double quotes around the text:
- Examples:
 - "This is a string literal."
 - "123 Main Street"
 - "x"
- Every character string is an object in Java, defined by the `String` class
- Every string literal represents a `String` object

There are many `String` objects but only one `String` class

Character Strings

- Every character string is an object in Java, defined by the `String` class
- Every string literal, delimited by double quotation marks, represents a `String` object
- The *string concatenation operator* (+) is used to append one string to the end of another
- It can also be used to append a number to a string
- A string literal cannot be broken across two lines in a program

Using Classes

- We invoke the `println` method to print a character string
- The `System.out` object represents a destination (the monitor screen) to which we can send output

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

object method name information provided to the method (parameters)

What about the “print” method?

String Concatenation

- The `+` operator is also used for arithmetic addition
- The function that it performs depends on the type of the information on which it operates
- If both operands are strings, or if one is a string and one is a number, it performs string concatenation
- If both operands are numeric, it adds them
- The `+` operator is evaluated left to right, but parentheses can be used to force the order

```
System.out.println ("24 and 45 concatenated: " + 24 + 45);
```

```
System.out.println ("24 and 45 added: " + (24 + 45));
```

Example

```

//*****
// Facts.java      Author: Lewis/Loftus
//
// Demonstrates the use of the string concatenation operator and the
// automatic conversion of an integer to a string.
//*****

public class Facts
{
    //-----
    // Prints various facts.
    //-----
    public static void main (String[] args)
    {
        // Strings can be concatenated into one long string
        System.out.println ("We present the following facts for your "
            + "extracurricular edification:");

        System.out.println ();

        // A string can contain numeric digits
        System.out.println ("Letters in the Hawaiian alphabet: 12");

        // A numeric value can be concatenated to a string
        System.out.println ("Dialing code for Antarctica: " + 672);

        System.out.println ("Year in which Leonardo da Vinci invented "
            + "the parachute: " + 1515);

        System.out.println ("Speed of ketchup: " + 40 + " km per year");
    }
}

```

Escape Sequences

- What if we wanted to print a the quote character?
- The following line would confuse the compiler because it would interpret the second quote as the end of the string

```
System.out.println ("I said "Hello" to you.");
```

- An *escape sequence* is a series of characters that represents a special character
- An escape sequence begins with a backslash character (\)

```
System.out.println ("I said \"Hello\" to you.");
```

Escape Sequences

- Some Java escape sequences:

Escape Sequence	Meaning
<code>\b</code>	backspace
<code>\t</code>	tab
<code>\n</code>	newline
<code>\r</code>	carriage return
<code>\"</code>	double quote
<code>'</code>	single quote
<code>\\</code>	backslash

Variables

- A *variable* is a name for a location in memory
- A variable must be declared by specifying the variable's name and the type of information that it will hold

data type variable name

```
int total;
```

int count, temp, result;

Multiple variables can be created in one declaration

Variables

- A variable can be given an initial value in the declaration

```
int sum = 0;
int base = 32, max = 149; // note syntax...
```

- When a variable is referenced in a program, its current value is used

Assignment

- An ***assignment statement*** changes the value of a variable
`total = 55;`
- The **assignment operator** is the = sign
- The expression on the right is evaluated and the result is stored in the variable on the left
- The value that was in `total` is overwritten
- You can **only** assign a value to a variable that is **consistent** with the variable's declared type

Constants

- A **constant** is an identifier that is similar to a variable except that it holds one value while the program is active
- The compiler will issue an error if you try to change the value of a constant during execution
- In Java, we use the `final` modifier to declare a constant

```
final int MIN_HEIGHT = 69;
```

Note: constants are written in caps to distinguish themselves from other 'variables' whose values can change.

- give names to otherwise unclear literal values
- facilitates updates of values used throughout a program
- prevent inadvertent attempts to change a value
- (Discuss: final float RATE = 0.15; only change value...)

Primitive Data

- There are exactly eight primitive data types in Java
- Four represent integers:
 - byte, short, int, long (no fractions)
- Two represent floating point numbers:
 - float, double (contain decimals)
- One represents characters: `char`
- One represents boolean values: `boolean`
- All have different 'sizes' and 'ranges'.....

Numeric Primitive Data

- Sizes and Ranges of storable values below.
- Use size as 'appropriate' but if in doubt, be generous.

Type	Storage	Min Value	Max Value
byte	8 bits	-128	127
short	16 bits	-32,768	32,767
int	32 bits	-2,147,483,648	2,147,483,647
long	64 bits	$< -9 \times 10^{18}$	$> 9 \times 10^{18}$
float	32 bits	+/- 3.4×10^{38} with 7 significant digits	
double	64 bits	+/- 1.7×10^{308} with 15 significant digits	

Numeric Primitive Data

- Default: int is 32 bits; but 45L or 45l => long
- Default: for decimal data:
 - assumes all literals are type double.
 - To make 'float' → 45.6F or 45.6f
 - Can say, if desired, 45.6D or 45.6d, but unnecessary.

Characters

- A `char` variable stores a single character from the Unicode character set
- A character set is an ordered list of characters, and each character corresponds to a unique number
- The Unicode character set uses sixteen bits per character, allowing for 65,536 unique characters
- It is an international character set, containing symbols and characters from many world languages
- Character literals are delimited by single quotes:

'a' 'x' '7' '\$' ', ' '\n'

→ '7' is not equivalent to 7 is not equivalent to "7"

Characters

- The ASCII character set is older and smaller than Unicode, but is still quite popular
 - Has evolved to eight-bits per byte.
- → (`char` is a 'primitive data type'; `String` is a class)
 - Because `String` is a class, it has many methods (operations) that can be performed on `String` objects!
- The ASCII characters are a subset of the Unicode character set, including:

uppercase letters	A, B, C, ...
lowercase letters	a, b, c, ...
punctuation	period, semi-colon, ...
digits	0, 1, 2, ...
special symbols	&, , \, ...
control characters	carriage return, tab, ...

Boolean

- A `boolean` value represents a true or false condition
- A boolean also can be used to represent any two states, such as a light bulb being on or off
- The reserved words `true` and `false` are the only valid values for a boolean type

```
boolean done = false;
```

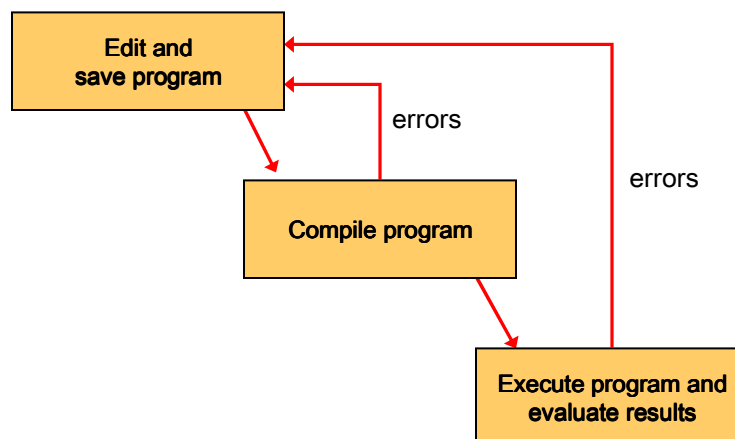
Compile-time and Run-time Errors

- An error identified by the compiler is called a `compile-time error`. (These are syntax errors)
 - If a compile-time error occurs, an executable version of the program is not created.
 - Use your editor to correct the error, then recompile your program.
 - DrJava will help you with this
- A `runtime error` causes the program to terminate **abnormally** during execution.
 - An example is an attempt to divide by zero.
 - In Java, many runtime errors are represented as `Exceptions`

Logical Errors

- **When your program has a logical error, it will compile and execute, but produces incorrect results. (These are runtime errors too!)**
 - A **logical error** occurs when a value is calculated incorrectly.
 - A programmer must test the program thoroughly, comparing the expected results to those that actually occur.
 - The process of finding and correcting defects in a program is called **debugging**.

Basic Program Development



Good Programming Practice

- Write a comment before each class, documenting the purpose of the class.
- Write end of line (single line) comment, documenting the purpose of the statement.
- Use DrJava to check your **syntax**.
- Declare each variable in each line, allowing end of line comment. E.g.

```
int age; // The age of my dog
```
- Choose meaningful variable names helps a program to be self-documenting. Easy to understand.

Summary

- **Java code is primarily interpreted so it can be used on any computer**
- **Java is based on objects which contain information (state) and actions (methods)**
- **Strings form an important part of data representations**
- **Assignments and basic data types allow for data storage and manipulation**
- **Reading: L+L: 1.5-1.6, 2.1-2.3**

