CSI1102
Introduction to Software Design

Chapter 1:
Introduction

Learning objectives

- Understand what problem solving entails
- Understand why problem solving skills are so important
- Describe the various levels of programming languages
- Understand a first Java program and its basic structure

Source: **Sections 1.3-1.5**, Chapter 1 (L&L)

A reminder:
Assumed background knowledge

Basic computer processing concepts

- Computer hardware components
  - CPU, I/O, main memory, secondary memory
- Digital computers and binary numbers
- Networks
  - Network connections
  - LAN and WAN
  - The Internet
  - The WWW

Read through Sections 1.0-1.2 to ensure you are "up to date"

So what is Problem Solving?

- The purpose of writing a program is **to solve a problem**
- The general steps in problem solving are:
  - Understand the problem
  - Dissect the problem into manageable pieces
  - Design a solution
  - Consider alternatives to the solution and refine it
  - Implement the solution
  - Test the solution and fix any problems that exist

Problem Solving:
"Divide and Conquer"

- Many software projects fail because the developer didn't really understand the problem to be solved
- We must **avoid assumptions** and **clarify ambiguities**
- As problems and their solutions become larger, we must organize our development into manageable pieces: **"Divide and Conquer"**

  → This technique is fundamental to software development

Object-oriented approach to Problem Solving

- We will **dissect our solutions into pieces** called **classes** and **objects**, taking an *object-oriented approach*
- *"If you want to eat an elephant, take one bite at a time".*

Problem solving through a programming language

- So suppose we have a problem to be solved
- We choose to design and implement a computer program to solve the problem
- → We use a programming language to encode the solution

7

Problem solving through a programming language

- A ***programming language*** specifies the words and symbols that we can use to write a program
- A programming language employs **a set of rules** that dictate how the words and symbols can be put together to form valid ***program statements***
- Examples of programming languages:
  - Fortran, Cobol, C++, C, Delphi, Pascal, Smalltalk and **JAVA**

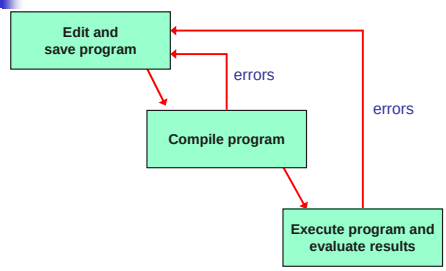
8

Different Programming Language Levels

- There are four programming language levels:
  - machine language
  - assembly language
  - high-level language
  - fourth-generation language
- Each type of CPU has its own specific *machine language*
- The other levels were created to make it easier for a human being to read and write programs

9

Basic Program Development



10

Problem solving using JAVA

- The Java programming language was created by Sun Microsystems, Inc.
- It was introduced in 1995 and it's popularity has grown quickly since
- It is an **object-oriented language**

11

Java Program Structure

- In the Java programming language:
  - A program is made up of one or more *classes*
  - A class contains one or more *methods*
  - A method contains program *statements*
- These terms will be explored in detail throughout the course, **starting from next week**
- A Java application **always** contains a method called `main`
- See `Lincoln.java` (page 30)

12

The Java Program Structure

```
//  comments about the class
public class MyProgram
{

}
```

class header

class body

Comments can be placed almost anywhere

13

The Java Program Structure (cont)

```
//  comments about the class
public class MyProgram
{
   //  comments about the method
   public static void main (String[] args)
   {

   }
}
```

method header

method body

14

A simple Java example

```
//*************************
// An example Java program
//
public class Hello
{
  //---------------------------------------
  // Task: Print the line "Hello World"
  //

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

}
```

15

About programming language: They all have the following in common

- Basic Components
  - Comments
  - Identifiers
  - (Reserved Words)
  - Symbols (e.g. <, >, =, ...)
  - White Spaces
- Syntax and Semantics
- Translation
- Errors

16

What are Comments?

- Comments in a program are called *inline documentation*
- They should be included to explain the purpose of the program and describe processing steps
- They do not affect how a program works
- Java comments can take three forms:

```
// this comment runs to the end of the line

/*  this comment runs to the terminating
    symbol, even across line breaks       */

/** this is a javadoc comment   */
```

17

The Importance of Comments

- Describe WHAT, WHY, HOW and by WHOM
- Very important for further use
  - To understand one year from now
  - To be able to develop and extend your program
- Do not add them at the end!!!!

18

What are Identifiers?

- *Identifiers* are the **words** a programmer uses in a program
- Sometimes we choose identifiers ourselves when writing a program (such as `Lincoln`)
- Sometimes we are using another programmer's code, so we use the identifiers that they chose (such as `println`)

19

Identifiers in Java

- An identifier can be made up of letters, digits, the underscore character (_), and the dollar sign
- Identifiers cannot begin with a digit
- Java is *case sensitive* - `Total`, `total`, and `TOTAL` are different identifiers
- By convention, Java programmers use different case styles for different types of identifiers, such as
  - *title case* for class names - `Lincoln`
  - *upper case* for constants - `MAXIMUM`

20

What are Reserved Words?

- Often we use special identifiers called *reserved words* that already have a predefined meaning in the language
- A reserved word cannot be used in any other way

E.g. **IF** (temp < 30) **THEN**

21

The Java Reserved Words

abstract	**else**	**interface**	**super**
boolean	**extends**	**long**	**switch**
break	**false**	**native**	**synchronized**
byte	**final**	**new**	**this**
case	**finally**	**null**	**throw**
catch	**float**	**package**	**throws**
char	**for**	**private**	**transient**
class	**goto**	**protected**	**true**
const	**if**	**public**	**try**
continue	**implements**	**return**	**void**
default	**import**	**short**	**volatile**
do	**instanceof**	**static**	**while**
double	**int**	**strictfp**	

22

About Syntax and Semantics

- The *syntax rules* of a language define how we can put together **symbols, reserved words**, and **identifiers** to make a valid program

E.g. if (height > tallest)
{
tallest = height;
}

- The *semantics* of a program statement define what that statement **means** (its purpose or role in a program)

E.g. We want to determine the tallest person.

23

About Syntax and Semantics

- A program that is syntactically correct is not necessarily logically (semantically) correct
- A program will always do what we tell it to do, not what we meant to tell it to do !!!
- Typical error → incorrect boundaries

E.g. (height > tallest) instead of (height >= tallest)

24

What are White Spaces?

- Spaces, blank lines, and tabs are called *white space*
- White space is used to separate words and symbols in a program
- Extra white space is ignored
- A valid Java program can be formatted in many ways
- Programs should be formatted to enhance readability, using consistent indentation
  - See `Lincoln2.java` (page 37)
  - See `Lincoln3.java` (page 38)

25

A BADLY structured Java example

```
//*************************
// A poorly formatted, though valid,  Java program
//
public class Hello {  public static void main (String[ ] args)
  { System.out.println ("Hello World"); }}
```

26

A nicely formatted example

```
//*****************************
// A nicely formatted Java program

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

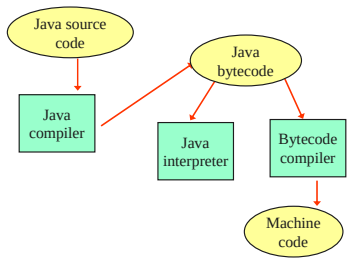
27

Translation of Programming Languages

- A program must be **translated into machine language** before it can be executed on a particular type of CPU
- 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* translates and executes one statement at a time
- The Java approach is somewhat different

28

Translation in Java



29

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 *architecture-neutral*

30

About Errors

A program can have three types of errors

- The compiler will find syntax errors and other basic problems (***compile-time errors***)
  - If compile-time errors exist, an executable version of the program is not created
- A problem can occur during program execution, such as trying to divide by zero, which causes a program to terminate abnormally (***run-time errors***)
- A program may run, but produce incorrect results, perhaps using an incorrect formula (***logical errors***)

31

Creating your program: Java Development Environments

- Sun Java Development Kit (JDK)
- Sun Forte for Java
- Borland JBuilder
- MetroWerks CodeWarrior
- Microsoft Visual J++
- Symantec Café
- Monash BlueJ
- RealJ (www.realj.com) ← Used in Lab B02

- Though the details of these environments differ, the basic compilation and execution process is essentially the same

32

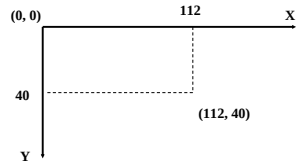
A word about Graphics

- Most computer programs have graphical components
- A picture or drawing must be digitized for storage on a computer
- A picture consists of pixels, and each pixel is stored separately
- Each pixel color is represented by a mix of Red, Blue and Green
- More details about graphics follow later in the course

33

Coordinate Systems

- Each pixel can be identified using a two-dimensional coordinate system
- When referring to a pixel in a Java program, we use a coordinate system with the origin in the top-left corner



34

Representing Color

- A black and white picture could be stored using one bit per pixel (0 = white and 1 = black)
- A colored picture requires more information; there are several techniques for representing colors
- For example, every color can be represented as a mixture of the three additive primary colors Red, Green, and Blue
- Each color is represented by three numbers between 0 and 255 that collectively are called an *RGB value*

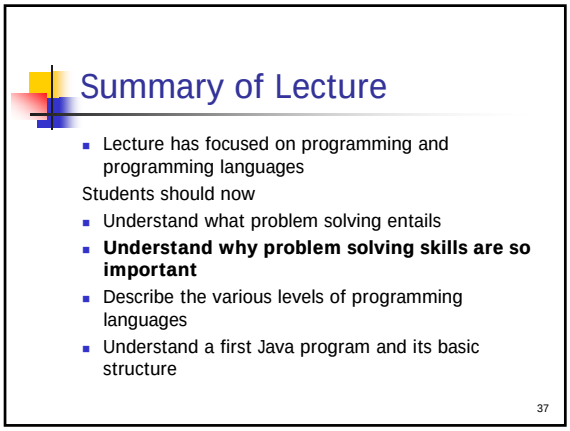
35

The Color Class

- A color in a Java program is represented as an object created from the `Color` class
- The `Color` class also contains several predefined colors, including the following:

Object	RGB Value
`Color.black`	`0, 0, 0`
`Color.blue`	`0, 0, 255`
`Color.cyan`	`0, 255, 255`
`Color.orange`	`255, 200, 0`
`Color.white`	`255, 255, 255`
`Color.yellow`	`255, 255, 0`

36


Summary of Lecture
Lecture has focused on programming and programming languages
Students should now
Understand what problem solving entails
Understand why problem solving skills are so important
Describe the various levels of programming languages
Understand a first Java program and its basic structure
37