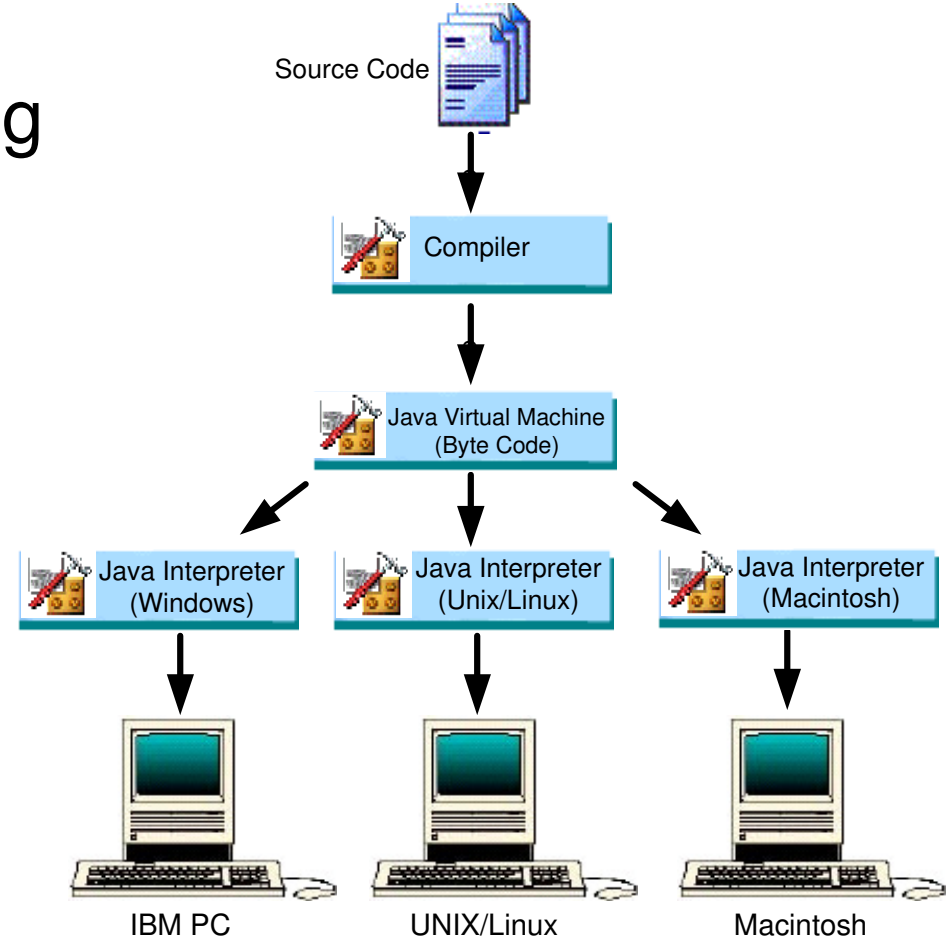
Chapter 2

Java Program Development

Review: Computer Systems & Java Programming

- Computer Systems
- Computer Programming
- Object-Oriented Programming
- Java Programming



Java Program Development

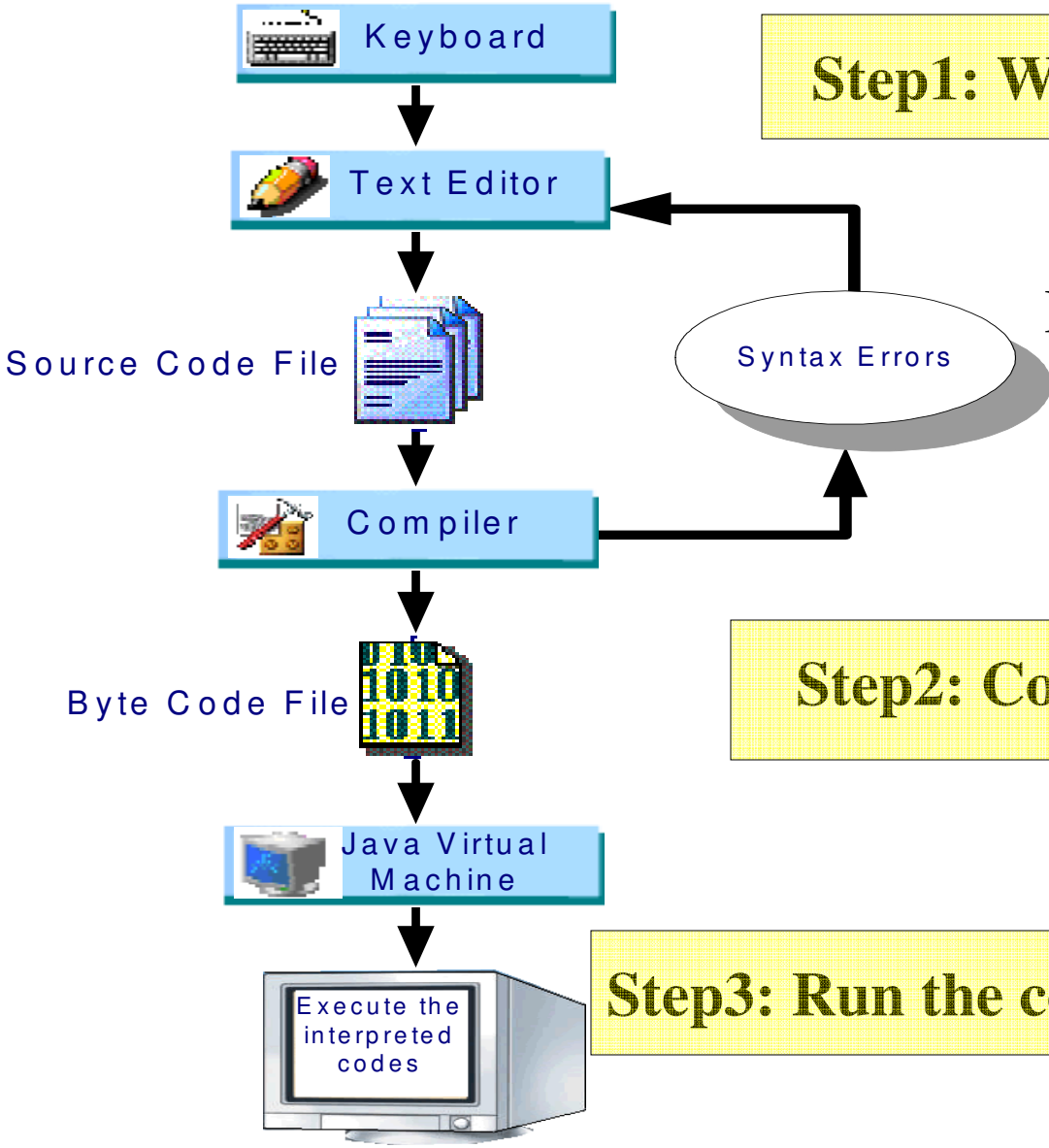
- **Development of a Java Program**
- Writing My First Java Program
- Program Development Process
- Problem Specification
- Problem Analysis
- Program Design
- Implementation
- Program Testing
- Documentation

Applets vs Java Applications

- Java Applets
  - Java programs to be downloaded via the Internet
  - Relatively small programs
  - Need a web browser
- Java Applications
  - Standalone programs
  - Generally larger applications

Only slight difference in programming,
this course focuses on Java applications

Steps to develop a Java Program



Step1: Write the Java source code

Step2: Compile the Java source code

Step3: Run the compiled Java code or class file

Java Development Environment (JDE)

(1) Use **Java Development Kit (JDK)** from Sun Microsystems (Version 1.5 or 1.6) – for compiler and class libraries. You may download the JDK 1.5 or 1.6 from Sun Microsystems: http:// java.sun.com/javase/downloads/index.jsp

(2) **Java Development Environment** (JDE) – user interface

- **JGRASP : http://www.jGRASP.com/**
- NetBeans
- Others: Jcreator, JBuilder, Eclipse

NB: In Labs – you will use the **Linux** operating System, and **standard editors** will be used for creating Java programs. However, you are free to use JDE in your lab or at home.

jGRASP

File: HelloStudents.java C:\Documents and Settings\Administrator\My Documents\SC Hui\Java course 2004\JAVA COURSE 2006-2007\Java Lecture...

File Edit View Build Project Settings Tools Window Help

All Files | Sort By Name

C:\Documents and Settings\Administrator\My Documer

- Average.class
- Average.java
- FindArray.class
- FindArray.java
- Histogram.class
- Histogram.java
- MatrixApp.class
- MatrixApp.java
- Reverse.class
- Reverse.java

Browse | Find | Debug | Workbench

```
1 /** A Java program
2 */
3 public class HelloStudents
4 {                                          // begin class
5   public static void main(String[] args)
6   {                                        // main method
7     System.out.println("Hello Students");  // statement
8   }                                        // end method
9 }
10
11
```

HelloStud...

Compile Messages | jGRASP Messages | Run I/O

Stop | Clear | Copy

Line:11 Col:1 Code:0 Top:1 OVS BLK

start | Lecture done | Chap01 | 2 Microsoft O... | File: HelloStud... | EN | 8:48 PM

Java Program Development

- Development of a Java Program
- **Writing My First Java Program**
- Program Development Process
- Problem Specification
- Problem Analysis
- Program Design
- Implementation
- Program Testing
- Documentation

My First Java Application Program

Step1: Write the Java source code in a text file

Filename: **HelloStudents**.java (must be ClassName.java)

```
/*
   Purpose:     A sample program to print a message
                "Hello Students!" On the screen.
   Author:      S.C. Hui
   Date:        12 July 2007
*/
public class HelloStudents {                        // begin class
  public static void main( String[] args ) {        // main method
     System.out.println( "Hello Students!" ); // statement
  }                                                 // end method
}                                                   // end class
```

Comments (documentation)

Three ways to write comments:

- **Compiler will ignore**
- **But help human to read**

(1)

/* multiple line comment
This is the first comment
This is the second comment
This is the third comment
*/ **remember to end**

(2)

// single line comment

Will mainly use these two types of comments

(3)

/** javadoc comment */ → Mainly used for documentation purpose

Syntax: Class Definition

Java program consists of at least one class.

```
public class HelloStudents {                    // begin class
  // class body
}                                               // end class
```

public class – reserved keywords

HelloStudents - class name

Class body is enlosed by { …}

Syntax: The main() Method

For every Java application, it must contain a **main()** method.
Java applications start execution from the main() method.

```
public class HelloStudents {                    // begin class
  public static void main(String[ ] args) {     // main method
    System.out.println("Hello Students!");      // statement
  }                                             // end method
}                                               // end class
```

Indentation!!

public static void – reserved keywords
main - method name
String[ ] args – method arguments
main method body is enclosed by { …}

Statements

A statement gives instruction to the computer.
A statement may be

- a ***simple*** statement
  - one statement terminated by a semicolon
- a ***compound*** statement
  - a sequence of one or more statements enclosed in **braces**. E.g.,

```
{
        statement_1;
        statement_2;
        ...
        statement_N;
}
```

Each of the statement_i can be a simple or a compound statement.

Using Methods from Other Classes

The HelloStudents.java program uses the method println() from the class System.out for displaying the text messages.

```
public class HelloStudents {                          // begin class
   public static void main(String[ ] args) {          // begin main method
      System.out.println("Hello Students!");          // statement
   }                                                  // end method
}                                                     // end class
```

Compiling My First Java Program

Step 2: To compile your program

Type:

$javac HelloStudents.java

where javac is JDK Java compiler

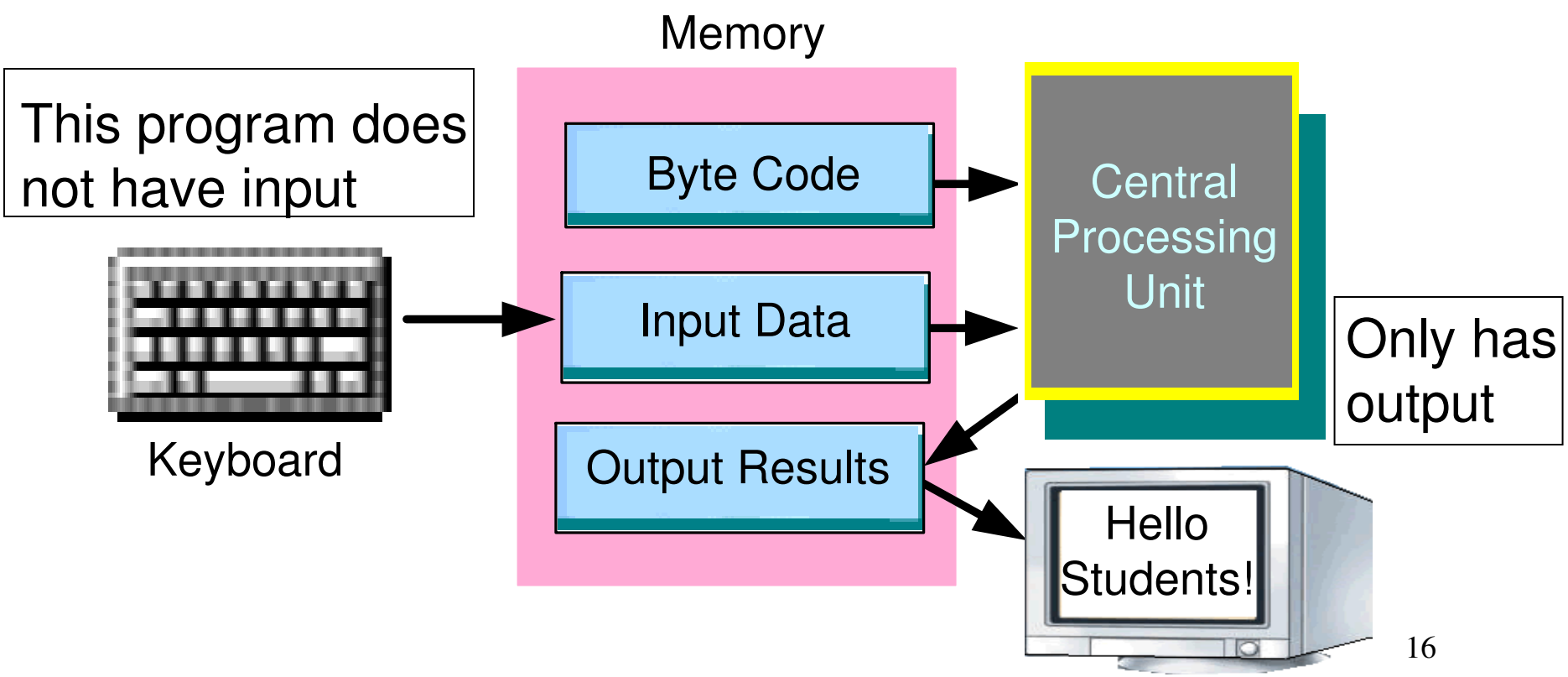
$ - command prompt

The compiler translates the source code into byte code and saves it into the file **HelloStudents.class**

Executing My First Java Program

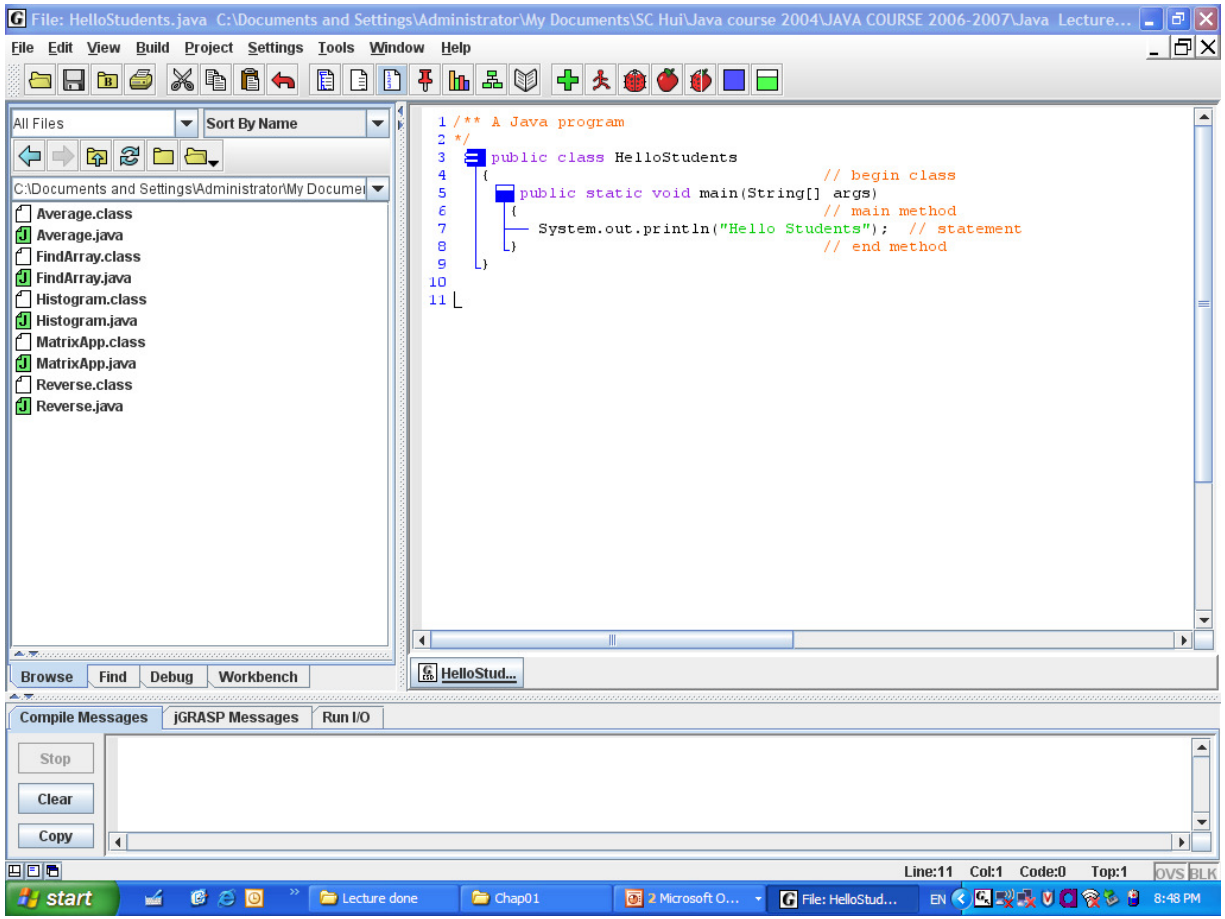
<u>Step 3: To run the byte code with the Java interpreter (JVM)</u>

Type: **$java HelloStudents**



Demo using jGRASP

Hello Students using Console I/O



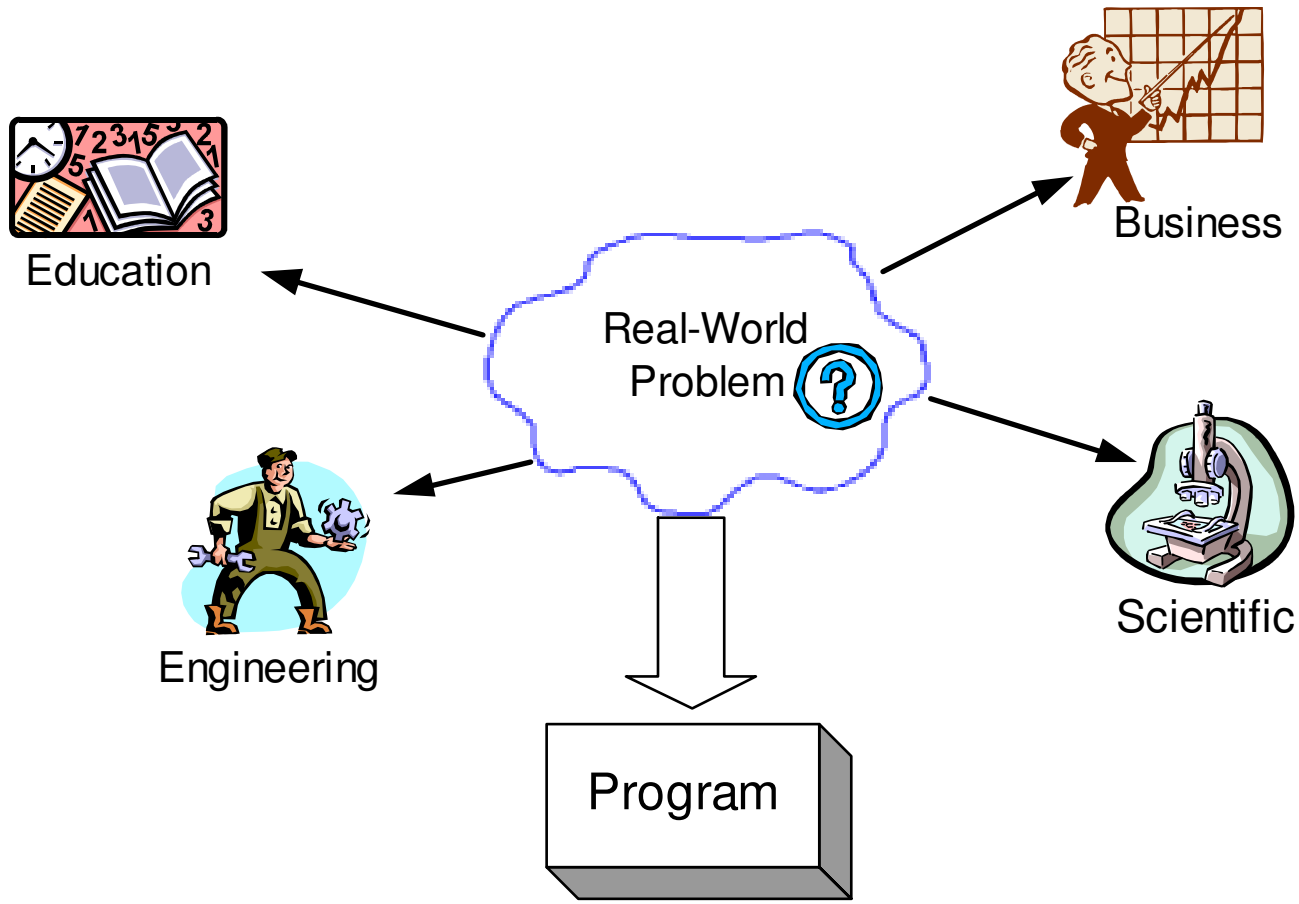
Java Program Development

- Development of a Java Program
- Writing My First Java Program
- **Program Development Process**
- Problem Specification
- Problem Analysis
- Program Design
- Implementation
- Program Testing
- Documentation

Problem! Problem! Problem!

- Some of your problems:
  - How to get through the examination?
  - How to get the degree?
  - How to get to know her/him?
  - …
- They specify a **goal**.
- They are blocked by lack of resources, knowledge, distance, time, etc.
- The process of finding a **solution** to reach the **goal** is called **Problem Solving.**

Solving Real-World Problems



Program Development Process

The program development process generally go through 5 phases:

1) **Problem Specification**
   - identify user requirements

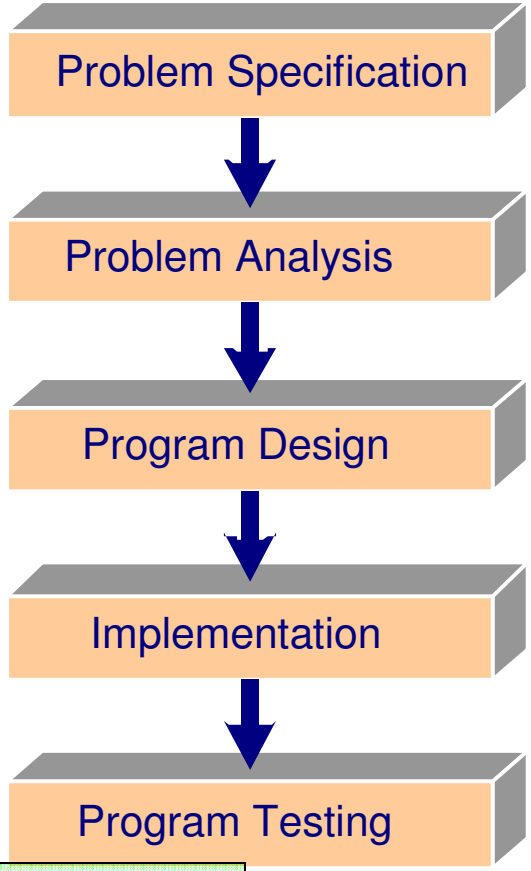
Logic

2) **Problem Analysis**
   - identify inputs, outputs, formulas
3) **Program Design**
   - write solution steps
     (using pseudocode or flowcharts)
   - go through dry-run to test the solution steps using test samples

4) **Implementation**
   - translate solution into computer program

Syntax

5) **Program Testing**
   - use test samples to test the program



Java Program Development

- Development of a Java Program
- Writing My First Java Program
- Program Development Process
  - **Problem Specification**
  - Problem Analysis
  - Program Design
  - Implementation
  - Program Testing
- Documentation

Problem Specification

This is a broad statement of the **user requirements**, in user terms. It sets

- the program **goals/purposes**
- the program **bounds** – the inputs and outputs

Example:

Design a program to solve the following problem:

"Design a **currency conversion program** that converts an input amount of US dollars into its equivalent amount in Singapore dollars. The program will read in the exchange rate from the user when computing the conversion."

Java Program Development

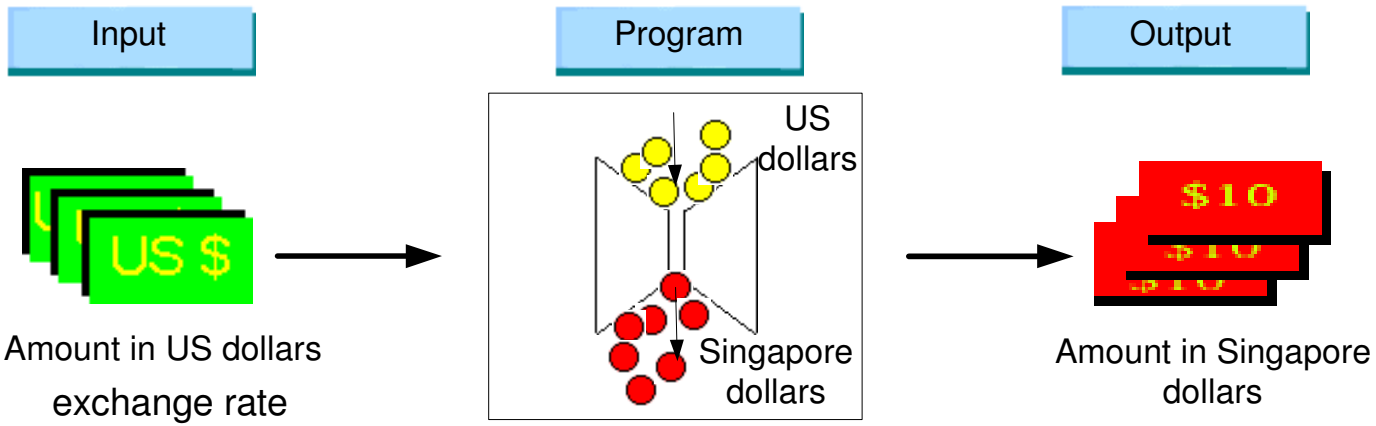
- Development of a Java Program
- Writing My First Java Program
- Program Development Process
  - Problem Specification
  - **Problem Analysis**
  - Program Design
  - Implementation
  - Program Testing
- Documentation

Problem Analysis

This phase produces a set of clear statements about the way the program is to work:

1. how the user uses the program (**program input**)
2. what **output** the program will generate
3. the **functionalities** (**formulae**) of the program
4. Additional requirements and **constraints**

Problem Analysis
Case Study



Singapore dollars = US dollars * exchange rate

***Required inputs*:**

- the amount in US dollars
- exchange rate from US dollar to Singapore dollar

***Required output*:**

- the equivalent amount in Singapore dollars

***Formulas*:**

- Singapore dollars = US dollars * exchange rate

Java Program Development

- Development of a Java Program
- Writing My First Java Program
- Program Development Process
  - Problem Specification
  - Problem Analysis
  - **Program Design**
  - Implementation
  - Program Testing
- Documentation

Program Design

Aim: design the **logic** of the program or **algorithm**.

Steps:

- Analyze the problem
- Design the program **logic** --- verified against system specification
- Design algorithm that gives the steps to transform the **inputs** into the intended **outputs**.

Algorithm

- Using **informal English** to describe the logic of the program
- Algorithm
  - must be unambiguous
  - every step must be clear and precise
  - specify the **order of steps** precisely

    [Sequence]
  - consider all possible **decision points**

    [Branching and Looping]
  - terminate in finite time
- Can be represented using

  (1) Pseudocode
  (2) Flowcharts

(1) Pseudocode

(sy-oooooooooooooo-doh! code)

A way to represent the logic or solution method of a program.

- no strict rules
- informal language - mix of **English** and **keywords**
- common **keywords**:
  IF, ELSE, ELSEIF, *WHILE, DO, ENDWHILE*
- other **keywords**:
  READ, PRINT, SET, INITILAIZE, COMPUTE, ADD, SUBTRACT, etc.

(2) Flowcharts

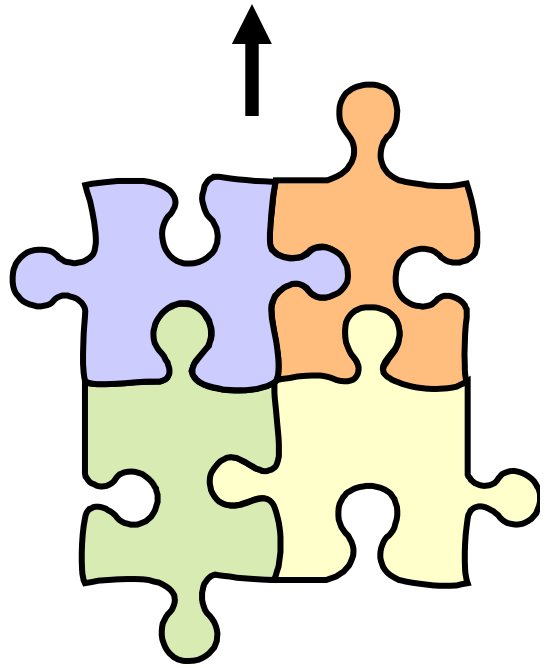
This is another way to represent the **logic** or **solution method** of a program by diagram. The flow of the control can be easily visualized.

Symbol	Name
	Process
	Decision
	Input / Output
	Terminal
	Flowlines

Basic Building Blocks for Solutions

Three basic **control structures**:

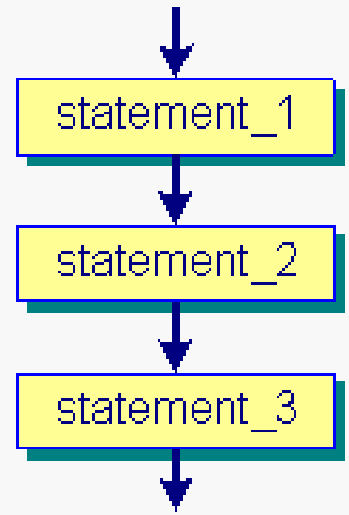
1. Sequence
2. Branching (Selection)
3. Looping (Repetition)



Sequence

- Do one thing at any one time (sequentially done)
- **E.g., Brush Teeth**
  1. Put tooth paste on tooth brush
  2. Brush teeth with tooth brush
- **E.g., breakfast**
  1. Toast bread
  2. Prepare coffee

- **Can the sequence be reversed?**
- Note: If there is one step **depending** on the other, the sequence is important
- Sequence Structure & its syntax (in Ch 3)

Pseudocode	Flowchart
statement_1; statement_2; statement_3;	

Selection

- We have to follow the **consequence** when we make a ***decision***
  - **If** I can sing well **then**

I will join talent competition

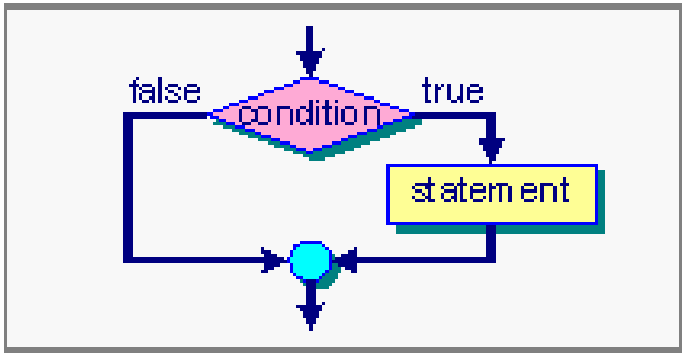
I will give big money to charity

  - **Else**

I will be a judge

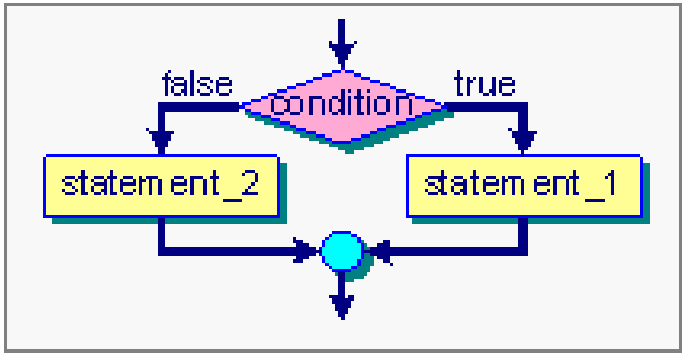
- Selection (or Branching) Structure & syntax (in Ch 5)

if structure

Pseudocode	Flowchart
IF condition statement; ENDIF	

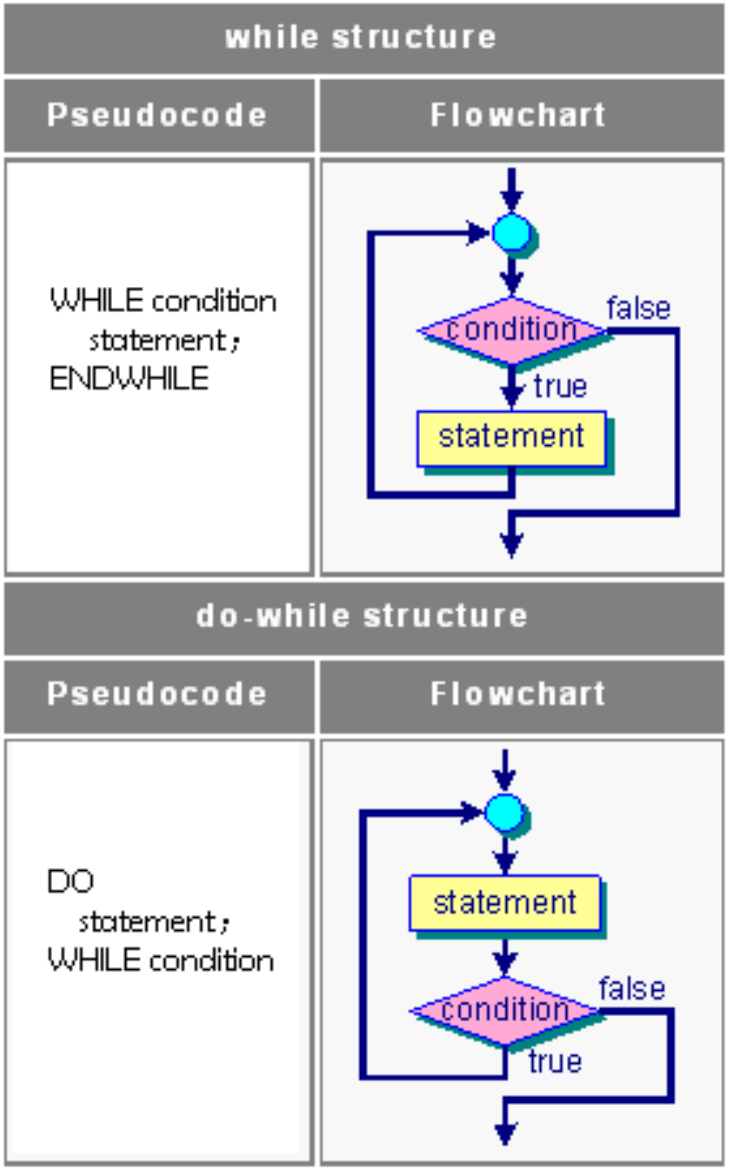
if-else structure

Pseudocode	Flowchart

IF condition
statement_1;
ELSE
statement_2;
ENDIF
	

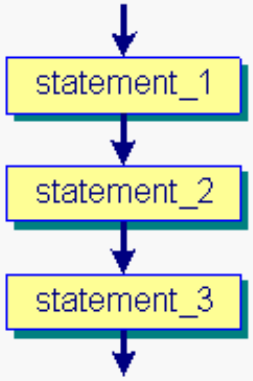
Repetition

- **For** everyone in this class
  - Get his/her height
  - Then Add the height together
- **At the end**, divide the total height by the number of students to get the average height.
- Repetition (or Looping) Structure & syntax (in Ch 6)

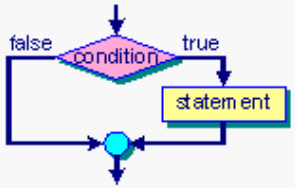


Which Representation?

Pseudocode | **Flowchart**

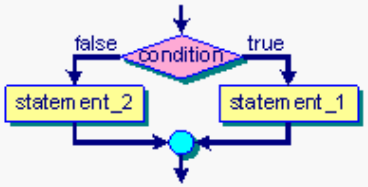
Pseudocode	Flowchart
statement_1; statement_2; statement_3;	

if structure

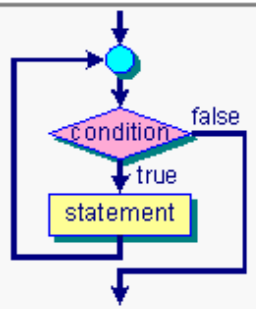
Pseudocode	Flowchart
IF condition statement; ENDIF	

if-else structure

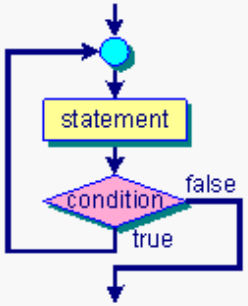
Pseudocode	Flowchart

IF condition
statement_1;
ELSE
statement_2;
ENDIF
	

while structure

Pseudocode	Flowchart
WHILE condition statement; ENDWHILE	

do-while structure

Pseudocode	Flowchart
DO statement; WHILE condition	

Pseudocode

We mainly use **Pseudocode** for program design. **Flowchart** will be used to illustrate the branching and looping concepts.

Rules for Pseudocode:

1. Write only *one statement* per line
2. *Capitalise* keywords
3. *Indent* to show hierarchy
4. *End* multi-line structures
5. Keep statements programming-language independent

Rules for Pseudocode

1. Example **keywords:**

READ	WRITE	PROMPT	OPEN	CLOSE
DISPLAY	ADD	SUBTRACT	MULTIPLY	DIVIDE
CALCULATE	INSERT	DELETE	REMOVE	APPEND
FIND	ACCESS	INDEX	CALL	ACTIVATE
DO	PERFORM			

2. Use **nouns** for **variable names**:

value	quantity	price	volume	salary	number
address	total	subtot	idNumber	payRate	cm2Inch

Program Design
Case Study

Initial Algorithm [Use informal English]

1. Get the amount of US dollars. *--- INPUTS*
2. Get the exchange rate.
3. Calculate the conversion. *--- USE FORMULA*
4. Print the converted amount in Singapore dollars. *- OUTPUT*

Algorithm in Pseudocode [Use keywords/variable names]

main:

LOGIC IN SEQUENCE

1 READ amtUsDollars, exchangeRate

2 COMPUTE conversion = amtUsDollars*exchangeRate

3 PRINT conversion

--- VARIABLE NAMES ??

It looks like program code
It feels like program code
But it isn't program code!!!

Program Dry-Run

Why do you want to dry-run your program?

- **Check** solution before you write program
- Concentrate on logic

Case Study:

Inputs: amtUsDollars = 100.0; exchangeRate = 1.75
conversion = 100.0 * 1.75 = 175.0
Output: 100.0 US dollars = 175.0 Singapore dollars

Program dry-run is a powerful technique for you to work out your solution without worrying about program syntax.

Thus, when you are ready, you are sure that your program would run with minimum effort in correcting it.

Java Program Development

- Development of a Java Program
- Writing My First Java Program
- Program Development Process
  - Problem Specification
  - Problem Analysis
  - Program Design
  - **Implementation**
  - Program Testing
- Documentation

Implementation

```
//*****************************************************************************
// Purpose:   To convert an input amount of US dollars into its equivalent
//            amount in Singapore dollars.
// Author:    S.C. Hui
// Date:      12 July 2007
//*****************************************************************************
import java.util.Scanner;
public class UsDollars
{
   public static void main(String[ ] args)
   {
      double amtUsDollars, amtSingDollars, exchangeRate;
      Scanner sc = new Scanner(System.in);    // for reading purposes
1     System.out.println("Enter the amount of US dollars:");
      amtUsDollars = sc.nextDouble();                    // using class Scanner
      System.out.println("Enter the exchange rate:");
      exchangeRate = sc.nextDouble();
2     amtSingDollars = amtUsDollars * exchangeRate;
3     System.out.println(amtUsDollars + " US dollars = " +
            amtSingDollars + " Singapore dollars");
   }
}
```

LOGIC IN SEQUENCE

Implementation

```
public class UsDollars
{
   public static void main(String[ ] args)
   {
      double amtUsDollars, amtSingDollars, exchangeRate;
```

LOGIC IN SEQUENCE

1

```
      // 1. READ amtUsDollars, exchangeRate
      Scanner sc = new Scanner(System.in);    // for reading purposes
      System.out.println("Enter the amount of US dollars:");
      amtUsDollars = sc.nextDouble();                    // using class Scanner
      System.out.println("Enter the exchange rate:");
      exchangeRate = sc.nextDouble();
```

2

```
      // 2. COMPUTE conversion = amtUsDollars*exchangeRate
      amtSingDollars = amtUsDollars * exchangeRate;
```

3

```
      // 3. DISPLAY results
      System.out.println(amtUsDollars + " US dollars = " +
            amtSingDollars + " Singapore dollars");
   }
}
```

Java Program Development

- Development of a Java Program
- Writing My First Java Program
- Program Development Process
  - Problem Specification
  - Problem Analysis
  - Program Design
  - Implementation
  - **Program Testing**
- Documentation

Program Testing

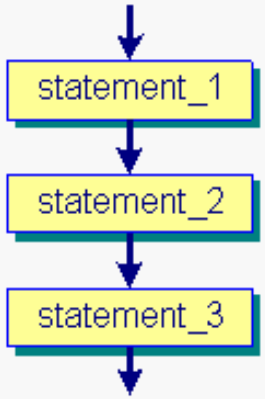
- A program is **correct** if it returns the correct result for every possible combination of input values.
- **Exhaustive testing**: use **all** possible combinations of input values and check the output is correct. This will take a whole year or forever to show the program is correct.

  -> **Impractical**

- What we can do: use **test data** that causes **every program path** (e.g. in branching and looping) to be executed at least once.

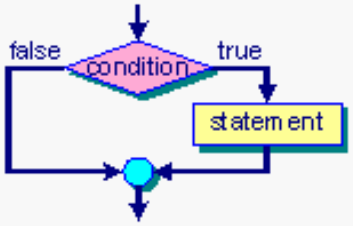
Testing Data for Program Testing

SEQUENCE

Pseudocode	Flowchart
statement_1; statement_2; statement_3;	

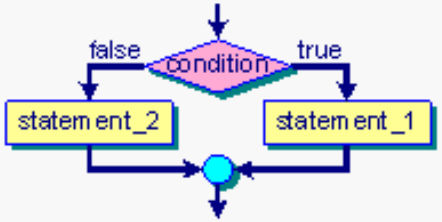
SELECTION

if structure

Pseudocode	Flowchart
IF condition statement; ENDIF	

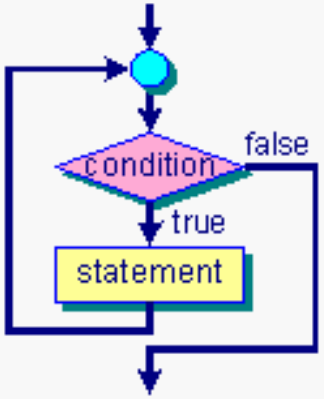
if-else structure

Pseudocode	Flowchart

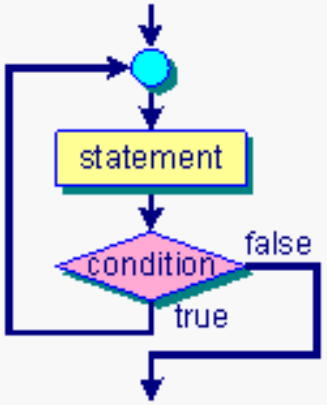
IF condition
statement_1;
ELSE
statement_2;
ENDIF
	

REPETITION

while structure

Pseudocode	Flowchart
WHILE condition statement; ENDWHILE	

do-while structure

Pseudocode	Flowchart
DO statement; WHILE condition	

Testing Results

Sample input and output

Enter the amount of US dollars:*100.0*

Enter the exchange rate:*1.75*

100.0 US dollars = 175.0 Singapore dollars

[One testing data should be sufficient, as only one single path from the logic sequence, you may use a calculator to verify the correctness of the execution]

Programming Errors

- **Syntax Errors**
  - “grammatical” errors
  - **detected by compiler**
  - found automatically
  - need to be fixed before the code can be compiled
  - error message may be misleading
  - e.g. *"UsDollars.java": ';' expected at line 10, column 58*

- **Runtime Errors**
  - execution error (e.g. divide by zero)
  - **detected during the execution of program**
  - error messages may be useful
  - sometimes not easy to fix

Programming Errors

- **Logic Errors**
  - due to error in designing the algorithm or implementation
  - **no compilation errors, no run-time error message**
  - most **difficult** to detect

Program Debugging

- The process of finding and correcting errors, especially **logic errors** (**BUG**!!!)
- Hand Tracing or Simulation
- **Program Tracing**
  - use System.out.println() at appropriate program locations
- **Debugger [Useful for more complex programs]**
  - Executing a single statement at a time
  - Tracing into or stepping over a method
  - Setting breakpoints
  - Displaying variables
  - Displaying call stacks
  - Modifying variables

Not understood yet?
Practice these with jGrasp

Program Tracing with Debugging Statements

```
public class UsDollars
{
   public static void main(String[ ] args) {
      …
      exchangeRate = sc.nextDouble();
      System.out.println("amtUsDollars = " + amtUsDollars);
      System.out.println("exchangeRate = " + exchangeRate);
      amtSingDollars = amtUsDollars * exchangeRate;
      …
   }
}
```

Print the input data for verification

Testing Results

Program input and output

Enter the amount of US dollars:*100.0*

Enter the exchange rate:*1.75*

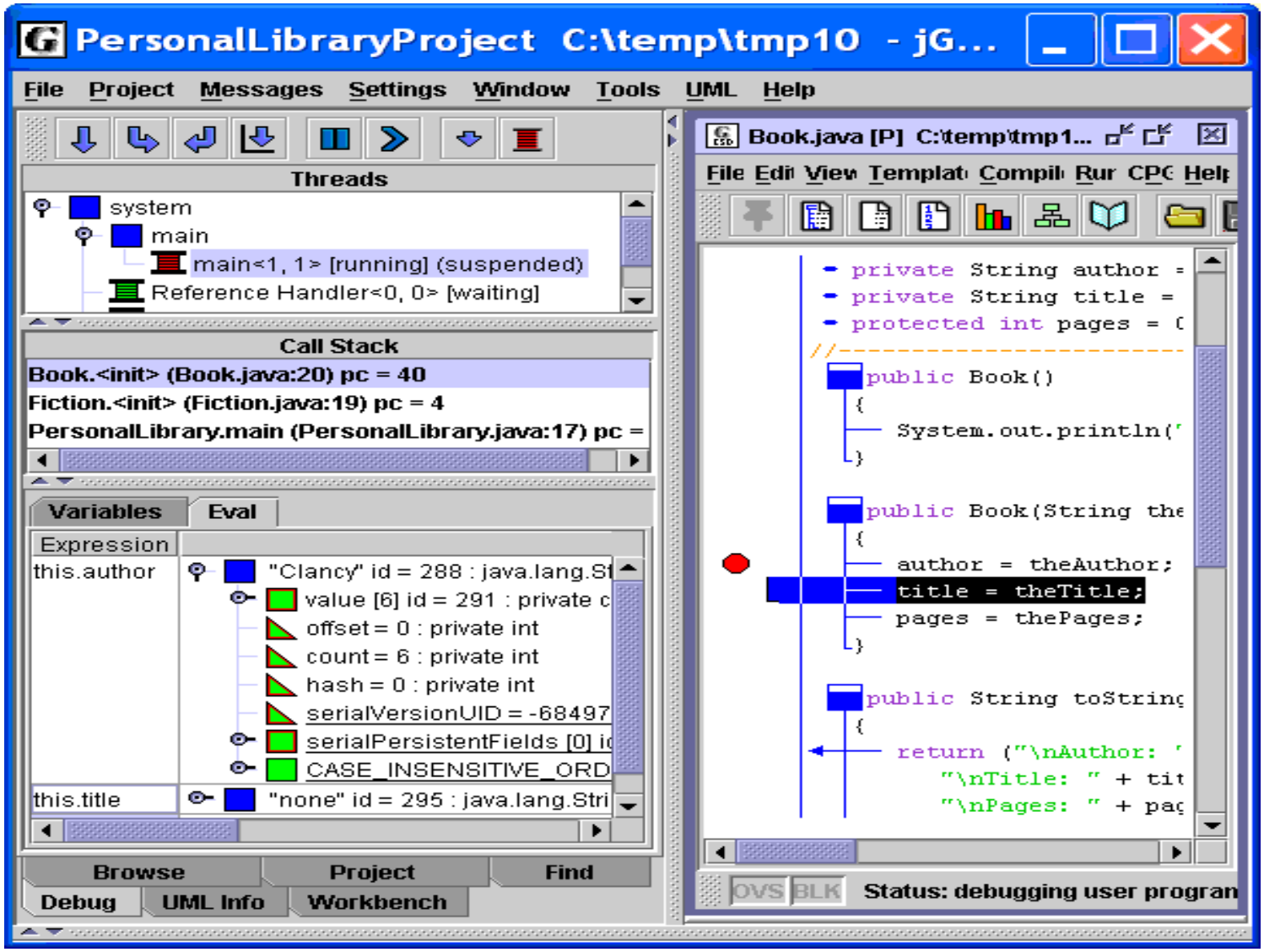
amtUsDollars = 100.0

exchangeRate = 1.75

100.0 US dollars = 175.0 Singapore dollars

jGRASP Debugger

Read http://www.jgrasp.org/tutorials187/06_Debugger.pdf



Java Program Development

- Development of a Java Program
- Writing My First Java Program
- Program Development Process
  - Problem Specification
  - Problem Analysis
  - Program Design
  - Implementation
  - Program Testing
- **Documentation**

Documentation

Documentation is needed for further modification and maintenance

<u>Proper documentation includes</u>:

- problem definition and specification;
- program inputs, outputs, constraints and mathematical equations;
- flowchart or pseudocode for the algorithm;
- source program listing;
- sample test run of the program; and
- user manual for end users.

Key Terms

- Java Development Environment (JDE)
- Java Development Toolkit (JDK)
- .java file, .class file
- java command, javac command
- Class definition
- main method, comment, statement
- program development process
- program specification
- program analysis
- program design
- algorithm
- pseudocode vs flowchart
- Control structures: sequence vs branching vs looping
- program implementation
- program testing
- programming errors
- syntax error vs runtime error vs logic error
- debugging
- program tracing vs debugger
- documentation

Review Questions

- What is *program development process*?
- What is the difference between *pseudocode* and *algorithm*?
- What is the difference between *pseudocode* and *Java source code*?
- What are the three types of *programming errors*?

Review Exercise

Problem Specification

Design a program to solve the following problem:

"Write a program to compute the area of a rectangle with sides *a* and *b*.
Remark: area of rectangle = *a***b*."

Problem Analysis

***Required inputs*: _____ and _____**
***Required output*: _____**
***Formulas*: _____**

Program Design

(1) Logic in sequence? (2) Variables?

Initial Algorithm: exercise

Algorithm in pseudocode:

main:

READ _____ and _____
COMPUTE _____
PRINT _____

Implementation

```
//*****************************************************************************
// Review Exercise
//*****************************************************************************
import java.util.Scanner;
public class ComputeRectangle
{
   public static void main(String[ ] args)
   {
      int _______, _________, _______;
      Scanner sc = new Scanner(System.in);              // for reading input
      // read user input
      System.out.println("Enter length a and width b :");
      __________ = sc.nextInt();
      __________ = sc.nextInt();
      // compute area
      __________ = _______________ * _________________;
      // print output
      System.out.println("The area of the rectangle is " + _______________);
   }
}
```

Further Reading

- Read Chapter 2 on “Java Program Development” of the textbook
- Read the “Tutorials for JGRASP Integrated Development Environment” – Available at http://www.jGRASP.com/
- Learn how to use the Debugging utility provided by jGRASP.