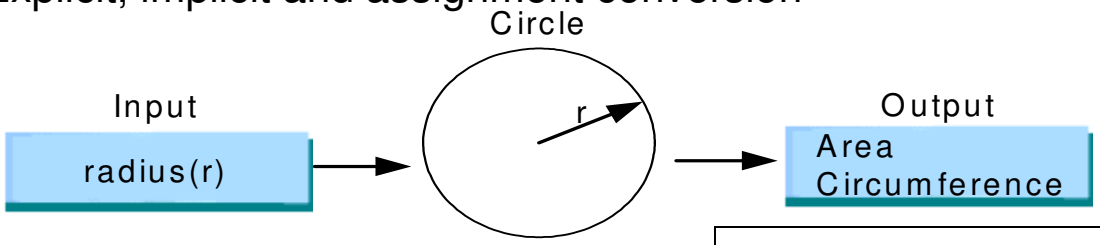
Chapter 4

Console Input/Output

Review: Data and Operators

- **Data Types**
  - Primitive data types (integer, floating–point, character, boolean)
  - Reference data types (e.g. Array, String)
- **Constants**
  - static final double PI = 3.14159;
- **Variables**
- **Operators and Expressions**
  - fundamental, assignment, increment/decrement, arithmetic, concatenation
- **Data Type Conversion**
  - Explicit, implicit and assignment conversion

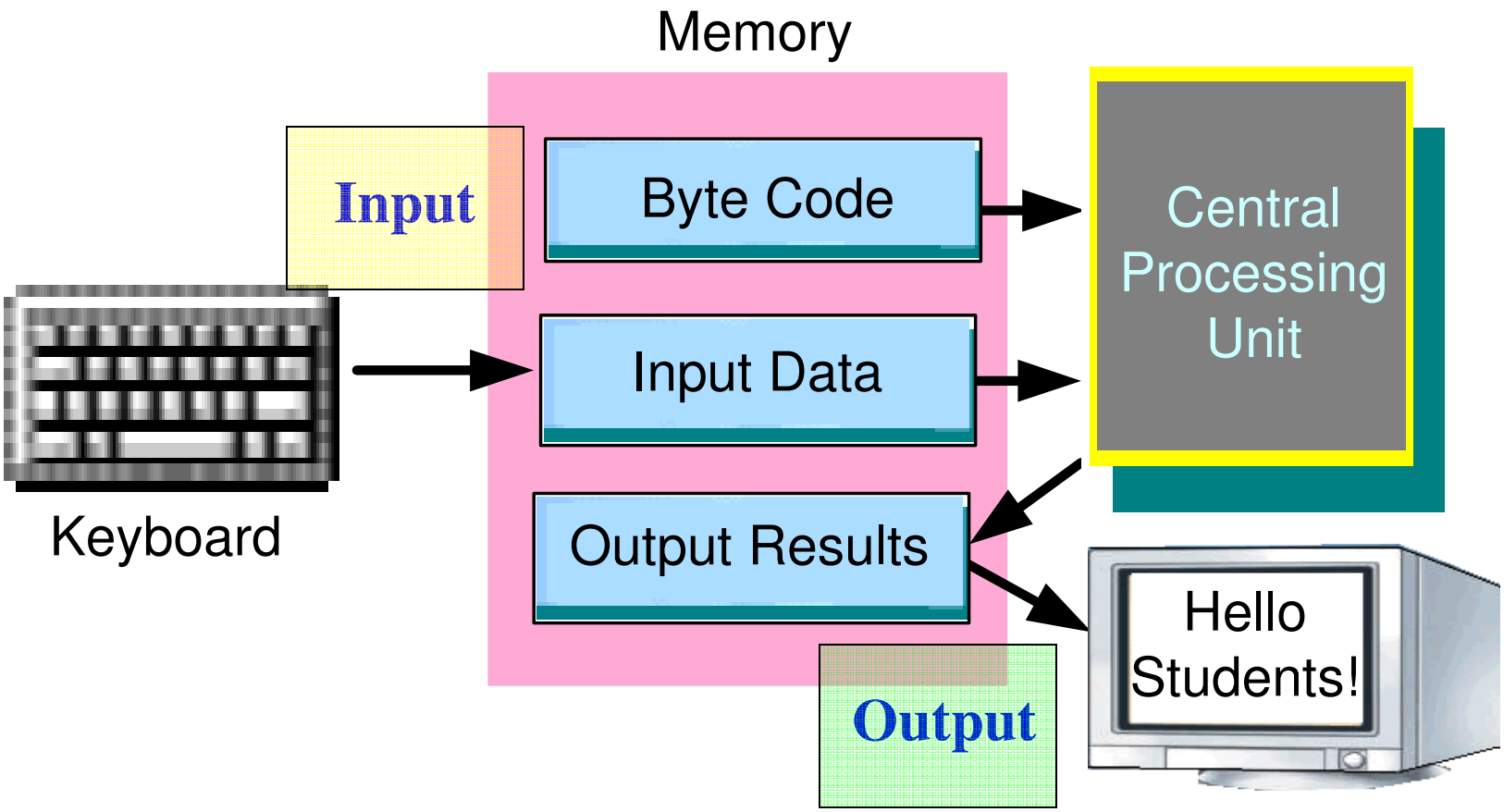



```
System.out.print("Enter the radius: ");
radius = sc.nextDouble();
```

```
System.out.println("The area is " + area);
System.out.println("The circumference is
    " + circumference);
```

```
area = PI * radius * radius;
circumference = 2 * PI * radius;
```

Review: Program Execution



Console Input/Output

- **Console Input/Output**
- Console Output: print() & println()
- Console Output: Using Message Dialog Box
- Importing Packages and Classes
- Formatted Console Output
- Console Input: Using the Scanner Class
- Console Input: Using Input Dialog
- Case Study

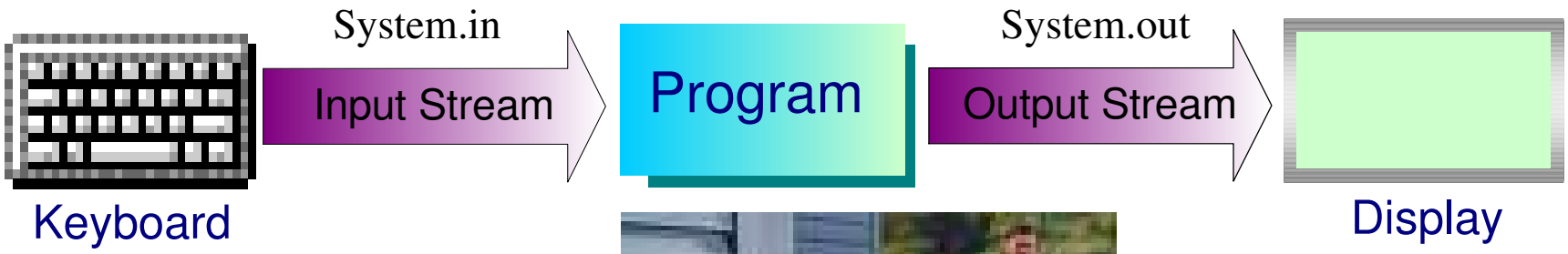
Console Input/Output

- **Input/output** (I/O) - the way a program communicates with the user.

Console Input/Output

Input from the keyboard

output to the screen



- In Java, there are special to perform console I/O:

```
- System.in:   standard input
- System.out: standard output
- System.err:  standard error
```

- **Console output**:
  - Use the stream System.out (provides the print(), printf() and println() methods).

- **Console input**:
  - Using the stream System.in is quite cumbersome.
  - Use the class **Scanner** that provides different input methods to read in different types of data.
  - The Scanner class is only available in Java 2 version 5 (or JDK version 1.5).

- May also use output/input **Dialog Box** (i.e. based on Graphical User Interface) [not required in this course]

Console Input/Output

- Console Input/Output
- **Console Output: print() & println()**
- Console Output: Using Message Dialog Box
- Importing Packages and Classes
- Formatted Console Output
- Console Input: Using the Scanner Class
- Console Input: Using Input Dialog
- Case Study

Console Output: print() and println()

- The **java.io.PrintStream** class contains two useful output **methods**:

- **print()**: prints the value of the argument in the specified data type
- **println()**: prints the value of the argument in the specified data type followed by a **new line**

- The general syntax:

```
- System.out.print(Output [+ Output]);
- System.out.println(Output [+ Output]);
```

where Output can be any data type of String, char, int, float or double. […] may be repeated zero or more times, but must be separated by **+**.

Example: Console Output

```
public class ConsoleOutput {
    public static void main(String[ ] args){
        System.out.print("Hello ");
        System.out.println("Students!");
        System.out.println("Hello\nStudents!");
        System.out.println("Hello\tStudents!");
    }
}
```

Program output:

Hello Students!
Hello
Students!
Hello Students!

Special characters such as \t, \\, \r, etc. can be used to separate the contents of the output string.

Example: Console Output

```
public class ConsoleOutput1 {
    public static void main(String args[ ]) {
     String name = "Phua Chu Kang";
     int age = 35;
     char gender = 'M';
     double height = 1.78;
     System.out.println("Hello! You are " + name);
     System.out.println("Very nice name");
     System.out.println("Your age = " + age);
     System.out.println("Your gender = " + gender);
     System.out.println("Your height = " + height);
    }
}
```

Program output:

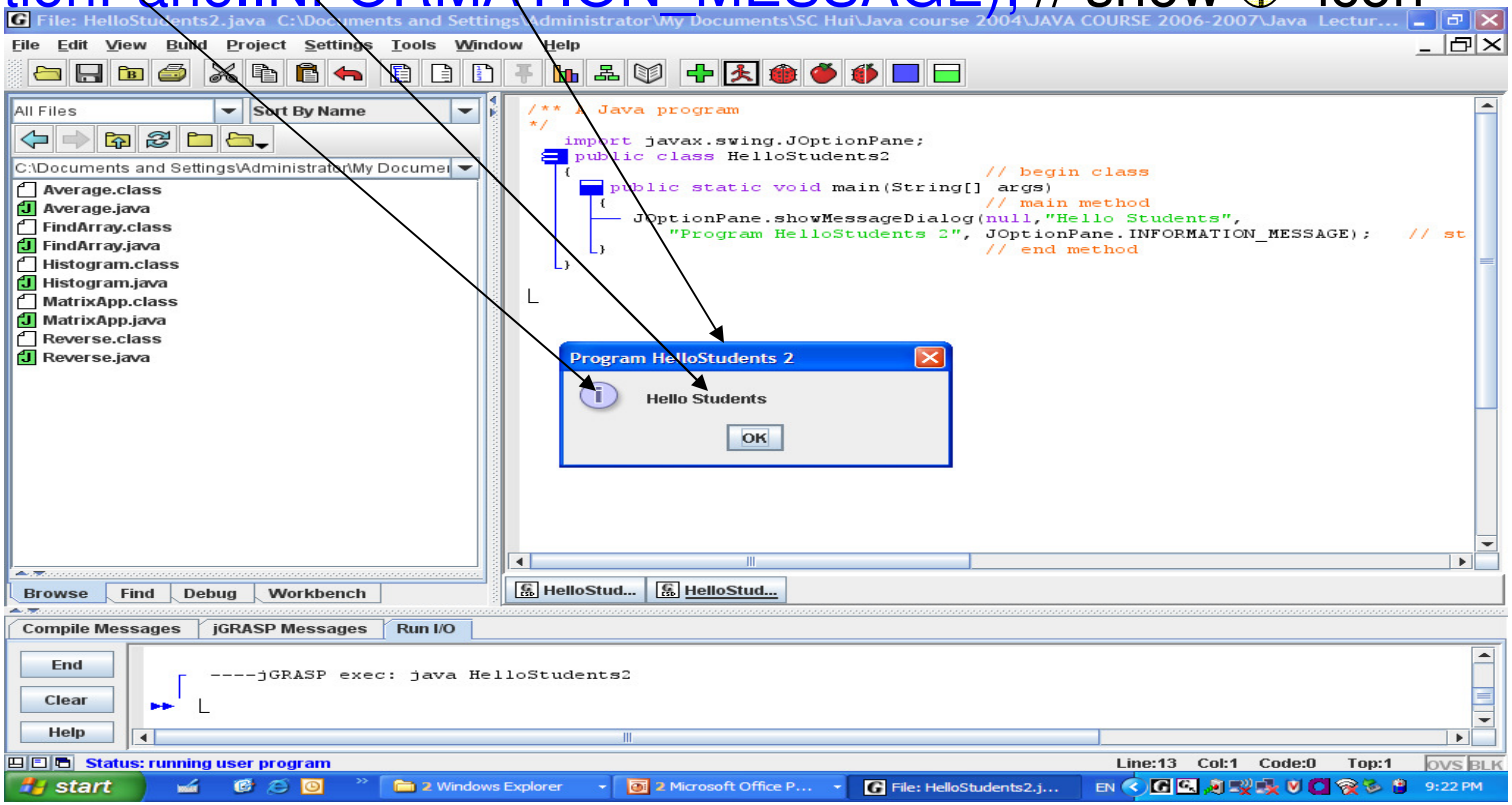
Hello! You are Phua Chu Kang
Very nice name
Your age = 35
Your gender = M
Your height = 1.78

Console Input/Output

- Console Input/Output
- Console Output: print() & println()
- **Console Output: Using Message Dialog Box**
- Importing Packages
- Formatted Console Output
- Console Input: Using the Scanner Class
- Console Input: Using Input Dialog
- Case Study

Console Output: Using Message Box

```
import javax.swing.JOptionPane;
public class UsingMessageBox {
    public static void main(String[] args) {
        JOptionPane.showMessageDialog( null, // always null
        "Hello Students!",                   // message
        "Program Hello Students 2",          // title
        JOptionPane.INFORMATION_MESSAGE); // show icon
    }
}
```



Console Input/Output

- Console Input/Output
- Console Output: print() & println()
- Console Output: Using Message Dialog Box
- **Importing Packages and Classes**
- Formatted Console Output
- Console Input: Using the Scanner Class
- Console Input: Using Input Dialog
- Case Study

Packages

- Packages are used to **group** classes. For example,

```
package test.mypackage;
public class Class1 {
    public static double method1() {…}
}
```

- The package name informs the compiler the **path name** for the **directory** containing the classes in the package.

- Assuming classes with test.mypackage are stored in c:\myjava\programs\test\mypackage, the class path can be set as:
  - In **Unix/Linux**: **setenv CLASSPATH .:/myjava/programs**
  - In **Windows**: **CLASSPATH=.;c:\myjava\programs**

Env. Variables in OS

The **dot** indicates the current directory.

Java Packages

Package	Description
`java.applet`	***Java Applet Package*** - for supporting applets.
`java.awt`	***Java Abstract Windowing Toolkit Package*** - for supporting graphical user interfaces in Java 1.0 and 1.1.
`java.awt.event`	***Java Abstract Windowing Toolkit Event Package*** - for event handling for graphics programming.
`java.beans`	***Java Beans Package*** - related to developing *beans*, components based on the JavaBeans architecture.
`java.lang`	***Java Language Package*** - contains the core Java classes such as **`Object`**, **`String`**, **`System`**, **`Math`**, **`Number`**, **`Character`**, **`Boolean`**, **`Short`**, **`Integer`**, **`Long`**, **`Float`** and **`Double`**. **- automatically imported** by the compiler to every Java program.
`java.io`	***Java Input/Output Package*** - for input and output streams and files.

Package	Description
`java.net`	***Java Networking Package*** - for network communications.
`java.rmi`	***Java Remote Method Invocation Package*** - for distributed computing.
`java.security`	***Java Security Package*** - for the security framework.
`java.sql`	***Java SQL Package*** - for accessing and processing data stored in databases.
`javax.swing`	***Java Swing GUI Components Package*** - for Java's Swing GUI components.
`javax.swing.event`	***Java Swing Event Package*** - for handling events for GUI components in the **`javax.swing`** package.
`java.text`	***Java Text Package*** - for manipulating information such as DecimalFormat, date, numbers, characters and strings. - also provides many internationalization capabilities.
`java.util`	***Java Utilities Package*** - contains many utilities such as Scanner, date, calendar, locale, vectors, stacks and hashing.

The import Statement

To access **a class** from a package of the Java API, we use the import statement.

```
import Package_Name.Class_Name;
```

For example:

```
import java.io.IOException;
```

We can also import **all classes** from a package:

```
import java.io.*;
```

or

```
import test.mypackage.*;
```

- Place the **import** statement at the beginning of the class file.
- No need to use import for **java.lang** package that contains **System** and **String** classes.

Notes on Package

- In Java, **every class belongs to a package**.
- If we do not specify any packages when we define the classes, then the classes will be considered as part of the **default package**.

Console Input/Output

- Console Input/Output
- Console Output: print() & println()
- Console Output: Using Message Dialog Box
- Importing Packages and Classes
- **Formatted Console Output**
- Console Input: Using the Scanner Class
- Console Input: Using Input Dialog
- Case Study

Formatted Console Output

- Java supports **formatted output** with
  - DecimalFormat Class
  - NumberFormat Class
  - System.out.printf() Method

  [for Self-Reading]

- Examples (in printf):

	Conversion Specification	Output on Screen
(1)	**%d**	125
(2)	**%+6d**	□□+125
(3)	**%-6d**	125□□□

	Conversion Specification	Output on Screen
(1)	**%f**	12.345678
(2)	**%+11.5f**	□□+12.34568
(3)	**%-11.5f**	12.34568□□□
(4)	**%+12.3e**	□□+1.235e+01
(5)	**%-12.3e**	1.235e+01□□□

Console Input/Output

- Console Input/Output
- Console Output: print() & println()
- Console Output: Using Message Dialog Box
- Importing Packages and Classes
- Formatted Console Output
- **Console Input: Using the Scanner Class**
- Console Input: Using Input Dialog
- Case Study

Console Input: Using the Scanner Class

- The **Scanner** class
  - Available from Java 2 Ver 5.
  - Can also be used for reading data from other sources such as ***files***, and ***parsing strings*** into *tokens* and *words* (will be discussed in Chapter 11).
  - Comes from the package **java.util**.

- To use the Scanner class, we need to:

(1) Import the class: **import java.util.Scanner;**

(2) Create a Scanner object:

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

(3) Use the Scanner methods to read console input.

```
Variable_Name = sc.Input_Method_Name();
```

e.g. `int data = sc.nextInt();`

Scanner Class Input Methods

Method name	Description
`boolean nextBoolean()`	Returns the next input token as a **`boolean`** value.
`byte nextByte()`	Returns the next input token as a **`byte`** value.
`short nextShort()`	Returns the next input token as a **`short`** value.
`int nextInt()`	Returns the next input token as an **`int`** value.
`long nextLong()`	Returns the next input token as a **`long`** value.
`float nextFloat()`	Returns the next input token as a **`float`** value.
`double nextDouble()`	Returns the next input token as a **`double`** value.
`String next()`	Returns the next input token as a **`String`** value.
`String nextLine()`	Returns all input remaining on the current line as a **`String`** value. (reading everything till \n)
`boolean hasNext()`	Returns **`true`** if another token exists in the input.
`Pattern delimiter()`	Returns the pattern of delimiters that the **`Scanner`** object is currently using.
`Scanner useDelimiter()`	Sets the pattern of delimiters for the **`Scanner`** object. The argument is a **`String`** or a **`Pattern`**.

Example: Console Input

```
import java.util.Scanner;
public class ConsoleInput
{
    public static void main(String[] args)    {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter your name: ");
        String name = sc.nextLine();
        System.out.print("Enter your age: ");
        int age = sc.nextInt();
        System.out.print("Enter your gender: ");
        String gender = sc.next();
        System.out.print("Enter your height: ");
        double height = sc.nextDouble();
        System.out.println();
        System.out.println("Hello! ");
        System.out.println("Your name is " + name);
        System.out.println("Your age is " + age);
        System.out.println("Your gender is " + gender);
        System.out.println("Your height is " + height);
    }
}
```

Program Output

Enter your name: *Phua Chu Kang*
Enter your age: *35*
Enter your gender: *M*
Enter your height: *1.78*

Hello! Your name is Phua Chu Kang
Your age is 35
Your gender is M
Your height is 1.78

Other Console Input Methods

- Apart from reading a line of input data, the Scanner class also provides methods that perform console input without reading in the whole line.

- For example:

```
int data1 = sc.nextInt();
int data2 = sc.nextInt();
int data3 = sc.nextInt();
```

read user input data: 10 20 30 ***Separated by space***

separated by blank spaces and assign the data into the variables data1, data2, data3 respectively.

- If the user enters incorrect input data, then an **exception** (error) will be thrown and the program will be terminated.

Example: Console Input

```
import java.util.Scanner;
public class ConsoleInput1
{
    public static void main(String[ ] args)
    {
        int data1, data2, data3;
        double real1, real2;
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter three integers: ");
        data1 = sc.nextInt();
        data2 = sc.nextInt();
        data3 = sc.nextInt();
        System.out.print("The three integers are: ");
        System.out.println(data1 + " " + data2 + " " + data3);
        System.out.print("Enter two doubles: ");
        real1 = sc.nextDouble();
        real2 = sc.nextDouble();
        System.out.println("The two doubles are: ");
        System.out.print(real1 + " " + real2);
    }
}
```

Program Input/Output

Enter three integers: *5 10 15*
The three integers are: 5 10 15
Enter two doubles: *5.5 10.5*
The two doubles are: 5.5 10.5

Caution: When using the nextLine() Method

- Need to be careful when we use the **nextLine()** method after reading in **other primitive data** such as integers and doubles.

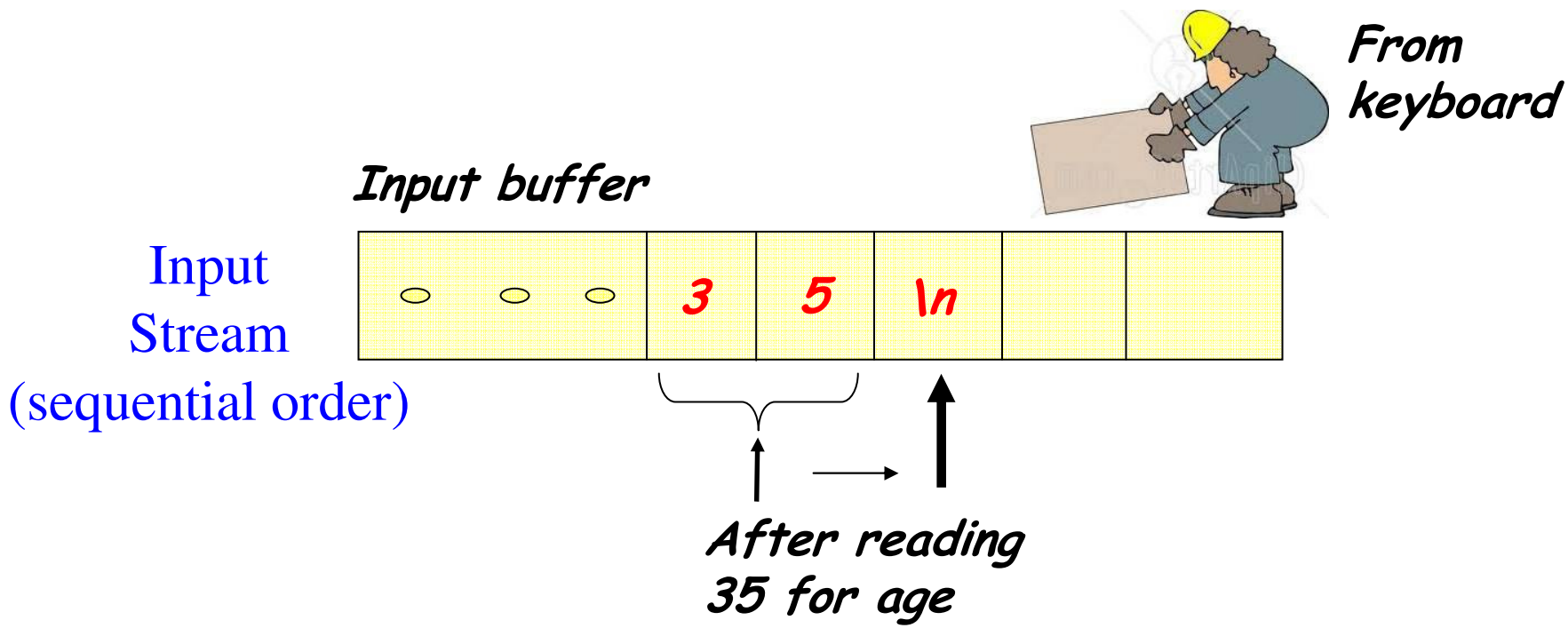
- **For example:**

```
System.out.print("Enter your age: ");
int age = sc.nextInt();
System.out.print("Enter your gender: ");
String gender = sc.nextLine();
System.out.print("Enter your height: ");
...
```

The corresponding output will be

```
Enter your name: Phua Chu Kang
Enter your age: 35
Enter your gender: Enter your height: 1.78
```

Caution: When using the nextLine() Method



- In this case, the nextLine() method reads the remainder of a line of text wherever the last keyboard reading left off.

Caution: When using the nextLine() Method

```
import java.util.Scanner;
public class UsingNextLine
{
  public static void main(String args[])
  {
    Scanner sc = new Scanner(System.in);

    System.out.print("Enter your name: ");
    String name = sc.nextLine();
    System.out.print("Enter your age: ");
    int age = sc.nextInt();

    // sc.nextLine();
    System.out.print("Enter your gender: ");
    String gender = sc.nextLine();
    System.out.print("Enter your height: ");
    double height = sc.nextDouble();

    System.out.println("Hello! ");
    System.out.println("Your name is " + name);
    System.out.println("Your age is " + age);
```

```
----jGRASP exec: java UsingNextLine

Enter your name: Hui Siu Cheung
Enter your age: 25
Enter your gender: Enter your height: 1.78
Hello!
Your name is Hui Siu Cheung
Your age is 25
Your gender is
Your height is 1.78

----jGRASP: operation complete.
```

Solution:

- Place another nextLine() method after reading in the age data in order to remove the remaining contents (i.e. '\n') in the **input buffer**.

```
...
int age
    = sc.nextInt();
sc.nextLine();
System.out.print(
"Enter your gender:
");
```

Console Input/Output

- Console Input/Output
- Console Output: print() & println()
- Console Output: Using Message Dialog Box
- Importing Packages and Classes
- Formatted Console Output
- Console Input: Using the Scanner Class
- **Console Input: Using Input Dialog**
- Case Study

Console Input: Using Input Dialogs

```
import javax.swing.JOptionPane;
public class ConsoleDialogInput
{
  public static void main(String args[])
  {

    String name = JOptionPane.showInputDialog("Enter your name: ");

    String ageString = JOptionPane.showInputDialog("Enter your age: ");
    int age = Integer.parseInt(ageString);

    String gender = JOptionPane.showInputDialog("Enter your gender: ");

    String heightString = JOptionPane.showInputDialog("Enter your height: ");
    double height = Double.parseDouble(heightString);

    String outpu                    ame + "\nYour age is " + age +
        "\nYour g                   is " + height;
    JOptionPane.                    ", JOptionPane.INFORMATION_MESSAGE);
  }
}
```

Input

Enter your name:

Phua Chu Kang

OK Cancel

INPUT DIALOG BOX

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

----jGRASP exec: java ConsoleDialogInput
```

Console Input: Using Input Dialogs

```
import javax.swing.JOptionPane;
public class ConsoleDialogInput {
  public static void main(String args[])     {
      String name = JOptionPane.showInputDialog("Enter your name: ");

      String ageString = JOptionPane.showInputDialog("Enter your age: ");
      int age = Integer.parseInt(ageString);

      String gender = JOptionPane.showInputDialog("Enter your gender: ");

      String heightString = JOptionPane.showInputDialog("Enter your height: ");
      double height = Double.parseDouble(heightString);

      String output = "Hello! " + "\nYour name is " + name + "\nYour age is " + age +
        "\nYour gender is " + gender + "\nYour height is " + height;
      JOptionPane.showMessageDialog(null, output, "Message",
       JOptionPane.INFORMATION_MESSAGE);
    }
  }
```

Console Input/Output

- Console Input/Output
- Console Output: print() & println()
- Console Output: Using Message Dialog Box
- Importing Packages and Classes
- Formatted Console Output
- Console Input: Using the Scanner Class
- Console Input: Using Input Dialog
- **Case Study**

Case Study
Calculating Salary

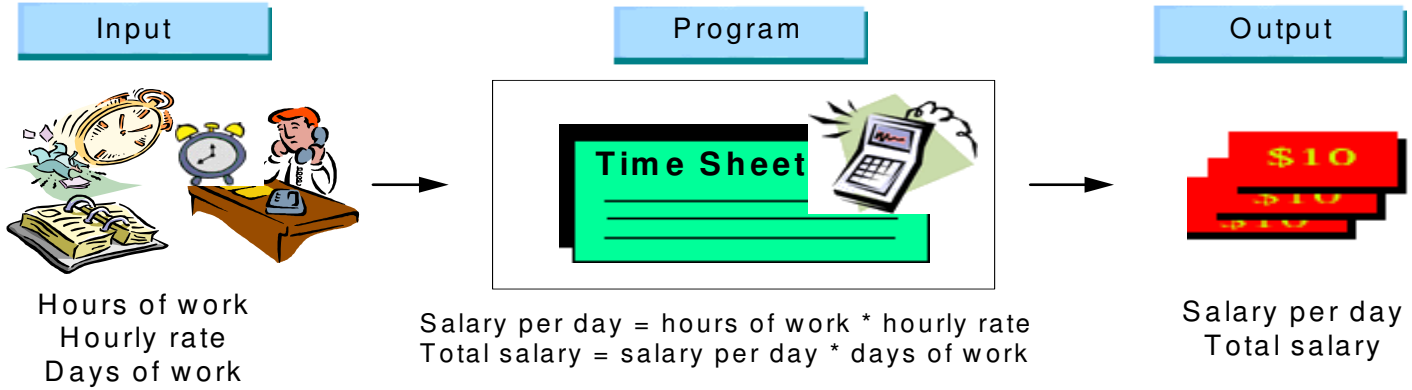
Problem Specification

"Write a program to calculate the monthly salary for an employee based on **input** information on

- the number of hours of work per day
- hourly rate
- number of days of work per month.

The program should **output** the salary per day and the total salary per month.
Assume that the number of hours of work per day is the same throughout the whole month . "

Problem Analysis



Required inputs:

- the employee number
- the number of hours of work
- the hourly rate
- the number of days of work

FORM VARIABLES ?

Required output:

- the salary per day
- the total salary per month

FORM VARIABLES ?

Formulas:

- salary per day = hours of work * hourly rate
- total salary = salary per day * days of work

Need to derive the formula

Program Design

Initial Algorithm

1. Read employee number.
2. Read number of hours of work, hourly rate, number of days of work.
3. Calculate salary per day.
4. Calculate total salary.
5. Print employee number, salary per day and total salary.

Algorithm in Pseudocode

LOGIC IN SEQUENCE

main:

```
READ empNo, hoursOfWork, hourlyRate, daysOfWork

COMPUTE salaryPerDay = hoursOfWork * hourlyRate
COMPUTE totalSalary = salaryPerDay * daysOfWork

PRINT empNo, salaryPerDay, totalSalary
```

Program Design

Program Dry-run

```
Inputs:
   empNum = 101416, hoursOfWork = 8,
   hourlyRate = 5.0, daysOfWork = 20

salaryPerDay = 8 * 5.0 = 40.0
totalSalary = 8 * 5.0 * 20 = 800.0

Outputs:
   Salary per day is $40.0
   Total salary is $800.0
```

Implementation

```
import java.util.Scanner;
public class CalculateSalary
{
   public static void main(String[] args)
   {
   int empNo;
   double hourlyRate, hoursOfWork, daysOfWork;
   double salaryPerDay, totalSalary;
   Scanner sc = new Scanner(System.in);
   // read input
   System.out.print("Enter employee number: ");
   empNo = sc.nextInt();
   System.out.print("Enter number of hours of work: ");
   hoursOfWork = sc.nextDouble();
   System.out.print("Enter hourly rate: ");
   hourlyRate = sc.nextDouble();
   System.out.print("Enter number of days of work: ");
   daysOfWork = sc.nextDouble();
```

VARIABLES

ASSIGNMENT STATEMENTS

```
    // compute results
    salaryPerDay = hoursOfWork * hourlyRate;
    totalSalary = salaryPerDay * daysOfWork;

    // print outputs
    System.out.println("The salary is: ");
    System.out.println("Employee no: " + empNo);
    System.out.println("Hours of work: " + hoursOfWork);
    System.out.println("Hourly rate: $" + hourlyRate);
    System.out.println("Days of work: " + daysOfWork);
    System.out.println("Salary per day: $" +
       salaryPerDay);
    System.out.println("Total salary: $" + totalSalary);
    }
}
```

Testing

Program input and output

Enter employee number: *101416*
Enter number of hours of work: *8*
Enter hourly rate: *5.0*
Enter number of days of work: *20*
Employee no: 101416
Hours of work: 8.0
Hourly rate: $5.0
Days of work: 20.0
Salary per day: $40.0
Total salary: $800.0

Key Terms

- System.out.println()
- System.out.printf()
- package
- import
- The Scanner class and the input methods

Further Reading

- Read Chapter 4 on “Console Input/Output” of the textbook

Additional Slides: Formatted Console Output

- Java supports **formatted output** with
  - DecimalFormat Class
  - NumberFormat Class
  - System.out.printf() Method

The DecimalFormat Class

- Java also provides two classes for formatted output: DecimalFormat and NumberFormat.

- In DecimalFormat Class - it provides a method called **format** in the class **java.text.DecimalFormat** – to display formats of **integers** and **floating point numbers**.

  e.g. $12.30

- To do this, there are three basic steps:

  (1) adding an import statement
  (2) creating the format string using the **new** operator
  (3) applying the format string to the numbers

Example: Formatted Output

```
import java.text.*;      // step 1
public class FormattedOutput {
    public static void main(String[] args) {
        DecimalFormat numForm = new
            DecimalFormat("000");          // step 2
        System.out.println(numForm.format(1));   // step 3
        System.out.println(numForm.format(12));
        System.out.println(numForm.format(123));
        System.out.println(numForm.format(1234));
    }
}
```

Creating an object

Format string

Program Output

001
012
123
1234

Format Specification

Symbol	Meaning
0	Used as a **compulsory** digit placeholder. If the digit is zero, it is automatically padded with the character 0.
#	Used as an **optional** digit placeholder. If the digit is zero, then it is displayed as absent and not as a space.
.	Used as a decimal placeholder.
,	Used as a grouping placeholder.
%	Used to express the number as a percentage. The number is multiplied by 100 and added a % sign.

Printing Formatted Integers

	Format String	Integer Number	Output on Screen
(1)	"000"	5	005
(2)	"000"	25	025
(3)	"000"	125	125
(4)	",000"	125	125
(5)	",000"	6125	6,125
(6)	"###"	5	5
(7)	"###"	25	25
(8)	"###"	125	125
(9)	",###"	125	125
(10)	",###"	6125	6,125

Example: Formatted Integer Output

```
import java.text.*;
public class FormattedOutput1 {

    public static void main(String[] args){
        int i = 123;
        short j = 123;
        long k = 123456789L;
        DecimalFormat numForm = new
          DecimalFormat("###");
        System.out.print("int i = ");
        System.out.println(numForm.format(i));
        System.out.print("short j = ");
        System.out.println(numForm.format(j));
        System.out.print("long k = ");
        System.out.println(numForm.format(k));
    }
}
```

Program Output

int i = 123
short j = 123
long k = 123456789

Printing Formatted Floating-point Numbers

	Format String	Number	Output on Screen
(1)	"000.00"	1234.6	1234.60
(2)	"000.##"	1234.6	1234.6
(3)	"###.00"	56.678	56.68
(4)	"###.##"	56.678	56.68
(5)	",000.00"	1234.6	1,234.60
(6)	",000.##"	234.6	234.6
(7)	",###.00"	3456.567	3,456.57
(8)	",###.##"	456.567	456.57

Example: Formatted Floating-point Output

```
import java.text.*;
public class FormattedOutput2 {
    public static void main(String[] args){
        float f = 1234.5f;
        double d = 12345.678;
        double p = 0.88689; // Note here
        DecimalFormat numForm = new
          DecimalFormat("$,###.00");
        System.out.print("float f = ");
        System.out.println(numForm.format(f));
        System.out.print("double d = ");
        System.out.println(numForm.format(d));
        DecimalFormat percentForm = new
          DecimalFormat("0.00%");
        System.out.print("percentage p = ");
        System.out.println(percentForm.format(p));
    }
}
```

Program Output

float f = $1,234.50
double d = $12,345.68
percentage p = 88.69%

The NumberFormat Class

- NumberFormat Class – provides generic formatting methods for **numbers** including

  (1) format() – for formatting according to object's pattern
  (2) getCurrencyInstance() – for specifying **currency** format
  (3) getPercentInstance() – for specifying **percentage** format

- Refer to Sun Microsystem website for more details

Formatted Output: printf()

- Java 2 Version 5 also provides a method **printf()** for formatted output.
- The format:

  System.out.printf(**control-string**, **argument-list**);

- The **control-string** is a string constant. It is printed on the screen.
- The **argument-list** contains a list of items such as item1, item2, …, etc.

Example:

```
System.out.printf(
"Age = %d; Gender = %c; Height = %f", age, gender, height
);
```

Program output:

Age = 35; Gender = M; Height = 1.780000

Formatted Output: printf()

- %x in the control-string is a conversion specification. An item will be substituted for it in the printed output.
- An item in the argument-list can be a constant, a variable or an expression like num1 + num2.
- The number of items must be the same as the number of conversion specifications.
- The type of the item must match the *conversionSpecifier*.

Note:

- The printf() method provided in Java is similar to C, so it aims for migration purposes.
- printf() does not have a clean object-oriented mind-set.
- For formatted output, we can also use **DecimalFormat** class.

Conversion Specifiers

Specifiers	Output
%b	a boolean value
%c	a single character
%d	a decimal integer
%o	an octal integer
%x	a hexadecimal integer
%f	a floating-point number, decimal notation
%e	a floating-point number, e-notation
%s	a string

Printing Integer Values

	Conversion Specification	Flag	Field Width	Conversion Specifier	Output on Screen
(1)	**%d**	none	none	**d**	125
(2)	**%+6d**	+	6	**d**	□□+125
(3)	**%-6d**	-	6	**d**	125□□□

- A ***flag*** is used to control the display of plus or minus sign of a number, and left or right justification. The **+** flag is used to print values with a plus sign if positive, and a minus sign otherwise. The **-** flag is used to print values left-justified.
- The ***field width*** gives the minimum field width to be used during printing. .

Printing Floating-point Values

	Conversion Specification	Flag	Field Width	Precision	Conversion Specifier	Output on Screen
(1)	**%f**	none	none	none	f	12.345678
(2)	**%+11.5f**	+	11	5	f	□□+12.34568
(3)	**%-11.5f**	-	11	5	f	12.34568□□□
(4)	**%+12.3e**	+	12	3	e	□□+1.235e+01
(5)	**%-12.3e**	-	12	3	e	1.235e+01□□□

- The ***precision*** field can be used for printing floating numbers. The precision field specifies the number of digits to be printed from the field width after the decimal point.