Chapter 2 Elementary Programming

Hu Zheng

huzheng@bupt.edu.cn

Exercise 01

1.1 包的使用：示例**DroidApp**。拷贝**./ch01/huz/**目录，构建带子目录的文件夹，了解其包含的子目录结构，并尝试用命令行方式进行带包编译和带包执行，从而掌握目录、包、**classpath**等概念的关系。

1.2 2*2线性方程组求解可以使用**Cramer**规则

$$ax + by = e$$
$$cx + dy = f$$

$$x = \frac{ed - bf}{ad - bc}$$

$$y = \frac{af - ec}{ad - bc}$$

编写程序，求解一下方程组并显示**x**和**y**的值。

$$3.4x + 50.2y = 44.5$$
$$2.1x + 0.55y = 5.9$$

```
Submission Deadline: the Next Monday(10/15/2018
  9:00am(class03) or 11:10am(class04)  )
```

Exercise 01

1.3　　财务应用：计算未来投资值

编写程序，读取投资总额、年利率和年数，然后使用下面的公式来显示未来投资金额：

未来投资金额=投资总额×（1+月利率）$^{年数×12}$

例如：如果输入的投资金额为**1000**，年利率为**3.25%**，年数为**1**，那么未来投资额为**1032.98**。

1.4　求出年数

编写程序，提示用户输入分钟数（例如十亿），然后显示这些分钟代表多少年和多少天。

为了简化问题，假设一年有**365天**。

1.5　　选做题，完善

https://github.com/huzhengatUCSD/Java_Course/blob/master/ch02/Calculator.java的未写好的方法，并试着编写一个

TestCalculator类，运用**Calculator**的方法完成对应运算功能。

```
//Submission Deadline: the Next Monday(10/16/2017
  9:00am(class03) or 11:10am(class04) )
```
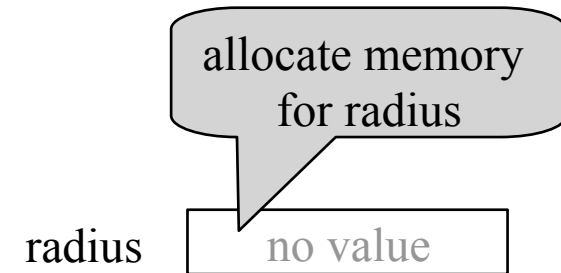
Introducing Programming with an Example

Ex 2.1 Computing the Area of a Circle

ComputeArea

Trace a Program Execution

allocate memory
for radius

```
public class ComputeArea {
  /** Main method */
  public static void main(String[] args) {
    double radius;
    double area;

    // Assign a radius
    radius = 20;

    // Compute area
    area = radius * radius * 3.14159;

    // Display results
    System.out.println("The area for the circle of radius
      " +
      radius + " is " + area);
  }
}
```
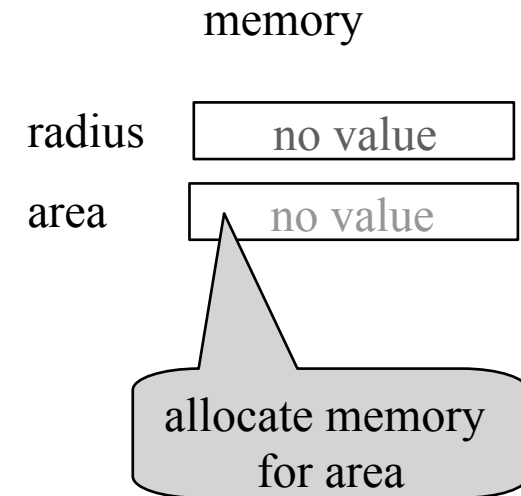
radius | no value

Trace a Program Execution

```
public class ComputeArea {
 /** Main method */
 public static void main(String[] args) {
   double radius;
   double area;

   // Assign a radius
   radius = 20;

   // Compute area
   area = radius * radius * 3.14159;

   // Display results
   System.out.println("The area for the circle of radius
     " +
     radius + " is " + area);
 }
}
```

memory

radius   no value

area   no value

allocate memory for area

6

Trace a Program Execution

```
public class ComputeArea {
  /** Main method */
  public static void main(String[] args) {
    double radius;
    double area;

    // Assign a radius
    radius = 20;

    // Compute area
    area = radius * radius * 3.14159;

    // Display results
    System.out.println("The area for the circle of radius
      " +
      radius + " is " + area);
  }
}
```

assign 20 to radius

radius | 20

area | no value

Trace a Program Execution

```
public class ComputeArea {
  /** Main method */
  public static void main(String[] args) {
    double radius;
    double area;

    // Assign a radius
    radius = 20;

    // Compute area
    area = radius * radius * 3.14159;

    // Display results
    System.out.println("The area for the circle of radius
      " +
      radius + " is " + area);
  }
}
```
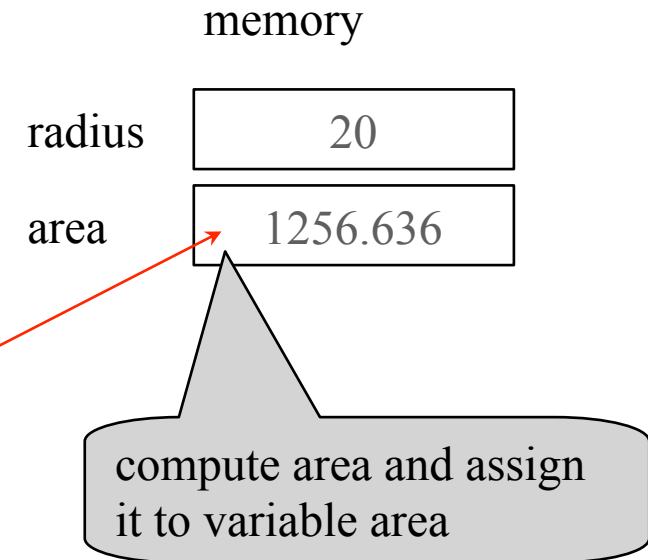
memory

radius | 20 |

area | 1256.636 |

compute area and assign it to variable area

8

Trace a Program Execution

```
public class ComputeArea {
  /** Main method */
  public static void main(String[] args) {
    double radius;
    double area;

    // Assign a radius
    radius = 20;

    // Compute area
    area = radius * radius * 3.14159;

    // Display results
    System.out.println("The area for the circle of radius
      " +
      radius + " is " + area);
  }
}
```

memory

radius | 20

area | 1256.636

print a message to the console

Reading Input from the Console

1. Create a Scanner object

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

2. Use the method nextDouble() to obtain to a double value. For example,

```
System.out.print("Enter a double value: ");
Scanner input = new Scanner(System.in);
double d = input.nextDouble();
```

ComputeAreaWithConsoleInput

ComputeAverage

Identifiers

✦ **An identifier is a sequence of characters that consist of letters, digits, underscores (_), and dollar signs ($).**

  ✦ **Character. isJavaIdentifierPart() returns `true`**

  ✦ **In Unicode Charsets: Characters bigger than 0xC0**

✦ **An identifier must start with a letter, an underscore (_), or a dollar sign ($). It cannot start with a digit.**

  ✦ **Character.isJavaIdentifierStart() returns `true`**

✦ **An identifier cannot be a reserved word.**

✦ **An identifier cannot be `true`, `false`, or `null`.**

✦ **An identifier can be of any length.**

■ **Correct Identifiers:**
  ◆ **Body, _test, $hello**

■ **Wrong Identifiers:**
  ◆ **5Test, hello*, world#, class**

Keywords

Some noteworthy points regarding Java keywords:

- **const** and **goto** are resevered words but not used.
- **true**, **false** and **null** are literals, not keywords.
- all keywords are in lower-case.

The following table shows the keywords grouped by category:

var is not a keyword, but rather an identifier with special meaning as the type of a local variable declaration

Category	Keywords
Access modifiers	private, protected, public
Class, method, variable modifiers	abstract, class, extends, final, implements, interface, native, new, static, strictfp, synchronized, transient, volatile
Flow control	break, case, continue, default, do, else, for, if, instanceof, return, switch, while
Package control	import, package
Primitive types	boolean, byte, char, double, float, int, long, short
Error handling	assert, catch, finally, throw, throws, try
Enumeration	enum
Others	super, this, void
Unused	const, goto

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);
```

Declaring Variables

```
int x;          // Declare x to be an
                // integer variable;

double radius;  // Declare radius to
                // be a double variable;

char a;         // Declare a to be a
                // character variable;
```

var is a type name, which was introduced in Java 10

Assignment Statements

```
x = 1;              // Assign 1 to x;
radius = 1.0;    // Assign 1.0 to radius;
a = 'A';            // Assign 'A' to a;
```

Declaring and Initializing in One Step

✦ `int x = 1;`

✦ `double d = 1.4;`

Named Constants

```
final datatype CONSTANTNAME = VALUE;

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

Naming Conventions

✦**Choose meaningful and descriptive names.**

✦**variables and method names:**
  – **Use lowercase. If the name consists of several words, concatenate all in one, use lowercase for the first word, and capitalize the first letter of each subsequent word in the name. For example, the variables `radius` and `area`, and the method `computeArea`.**
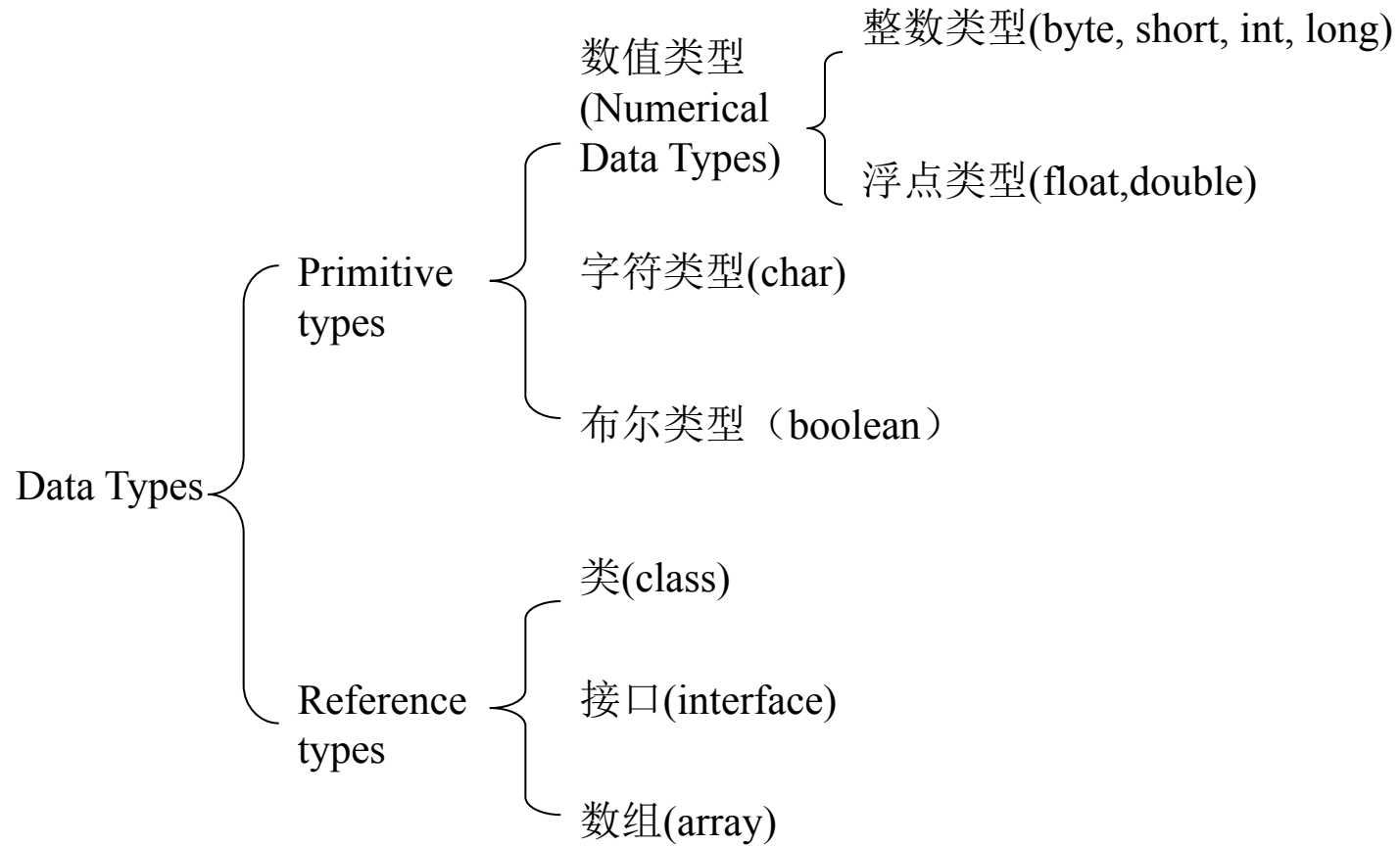
Naming Conventions, cont.

✦ **Class names:**
  – **Capitalize the first letter of each word in the name. For example, the class name `ComputeArea`.**

✦ **Constants:**
  – **Capitalize all letters in constants, and use underscores to connect words. For example, the constant `PI and` `MAX_VALUE`**

Data Types

数值类型
(Numerical
Data Types)

整数类型(byte, short, int, long)

浮点类型(float,double)

字符类型(char)

布尔类型（boolean）

Primitive
types

Data Types

类(class)

接口(interface)

数组(array)

Reference
types

Numerical Data Types

Name	Range	Storage Size
byte	-2^7 to $2^7 - 1$ (-128 to 127)	8-bit signed
short	-2^{15} to $2^{15} - 1$ (-32768 to 32767)	16-bit signed
int	-2^{31} to $2^{31} - 1$ (-2147483648 to 2147483647)	32-bit signed
long	-2^{63} to $2^{63} - 1$ (i.e., -9223372036854775808 to 9223372036854775807)	64-bit signed
float	Negative range: -3.4028235E+38 to -1.4E-45 Positive range: 1.4E-45 to 3.4028235E+38	32-bit IEEE 754
double	Negative range: -1.7976931348623157E+308 to -4.9E-324 Positive range: 4.9E-324 to 1.7976931348623157E+308	64-bit IEEE 754

There is NO UNSIGNED integer type in Java

Integer

- **Decimal：124，-100；**
- **Octal：0 as prefix, followed by 0~7 : ex. 0134；**
- **Hexadecimal：0x or 0X as prefix, followed by 0~9 or A~F。**

Reading Numbers from the Keyboard

```
Scanner input = new Scanner(System.in);
int value = input.nextInt();
```

Method	Description
nextByte()	reads an integer of the **byte** type.
nextShort()	reads an integer of the **short** type.
nextInt()	reads an integer of the **int** type.
nextLong()	reads an integer of the **long** type.
nextFloat()	reads a number of the **float** type.
nextDouble()	reads a number of the **double** type.

Numeric Operators

Name	Meaning	Example	Result
+	Addition	34 + 1	35
-	Subtraction	34.0 - 0.1	33.9
*	Multiplication	300 * 30	9000
/	Division	1.0 / 2.0	0.5
%	Remainder	20 % 3	2

Integer Division

+, -, *, /, and %

5 / 2 yields an integer 2.

5.0 / 2 yields a double value 2.5

5 % 2 yields 1 (the remainder of the division)

Problem: Displaying Time

Write a program that obtains minutes and remaining seconds from seconds.

DisplayTime

Floating-Point numbers

Calculations involving floating-point numbers are approximated because these numbers are not stored with complete accuracy.

calculations with integers yield a precise integer result.

double vs. float

The double type values are more accurate than the float type values. For example,

```
System.out.println("1.0 / 3.0 is " + 1.0 / 3.0);
```

S	e'(11bit)	M'(52bit)

```
System.out.println("1.0F / 3.0F is " + 1.0F / 3.0F);
```

S	e'(8bit)	M'(23bit)

Exponent Operations

```
System.out.println(Math.pow(2, 3));
// Displays 8.0
System.out.println(Math.pow(4, 0.5));
// Displays 2.0
System.out.println(Math.pow(2.5, 2));
// Displays 6.25
System.out.println(Math.pow(2.5, -2));
// Displays 0.16
```

Number Literals

A *literal* is a constant value that appears directly in the program.

int i = 34;

long x = 1000000;

double d = 5.0;

Integer Literals

An integer literal can be assigned to an integer variable as long as it can fit into the variable. A compilation error would occur if the literal were too large for the variable to hold.

An integer literal is assumed to be of the int type, whose value is between -2^{31} (-2147483648) to $2^{31}-1$ (2147483647).

Integer literal of the long type, append it with the letter L or l.

Floating-Point Literals

Floating-point literals are written with a decimal point. By default, a floating-point literal is treated as a double type value.

You can make a number a float by appending the letter f or F, and make a number a double by appending the letter d or D.

100.2f or 100.2F for a float number

100.2d or 100.2D for a double number

Scientific Notation

Floating-point literals can also be specified in scientific notation,

1.23456e+2, or 1.23456e2, is equivalent to 123.456
1.23456e-2 is equivalent to 0.0123456

Arithmetic Expressions

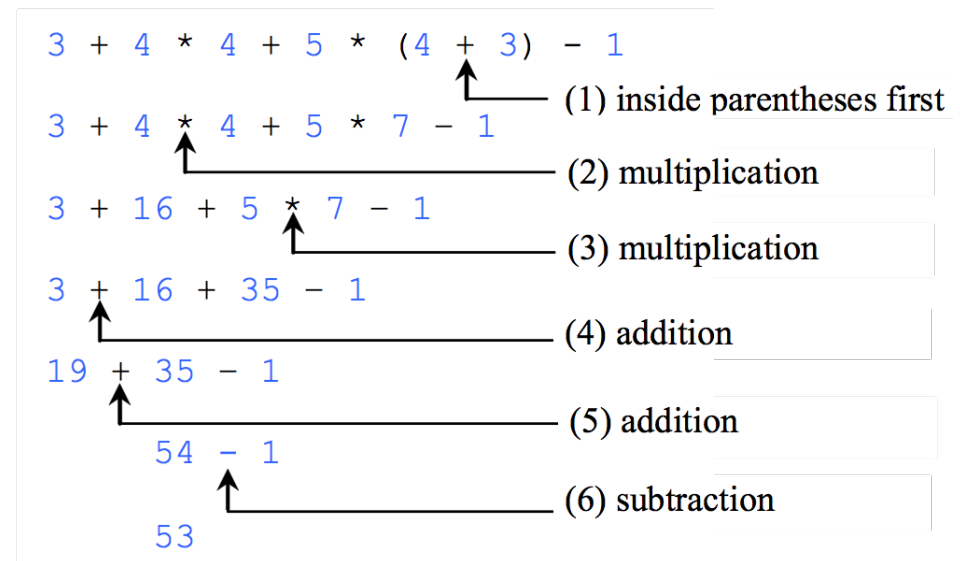$$\frac{3+4x}{5} - \frac{10(y-5)(a+b+c)}{x} + 9(\frac{4}{x} + \frac{9+x}{y})$$

is translated to

(3+4*x)/5 – 10*(y-5)*(a+b+c)/x + 9*(4/x + (9+x)/y)

How to Evaluate an Expression

You can safely apply the arithmetic rule for evaluating a Java expression.

The result of a Java expression and its corresponding arithmetic expression are the same.

```
3 + 4 * 4 + 5 * (4 + 3) - 1
```
——— (1) inside parentheses first
```
3 + 4 * 4 + 5 * 7 - 1
```
——— (2) multiplication
```
3 + 16 + 5 * 7 - 1
```
——— (3) multiplication
```
3 + 16 + 35 - 1
```
——— (4) addition
```
19 + 35 - 1
```
——— (5) addition
```
54 - 1
```
——— (6) subtraction
```
53
```

Problem: Converting Temperatures

Write a program that converts a Fahrenheit degree to Celsius using the formula:

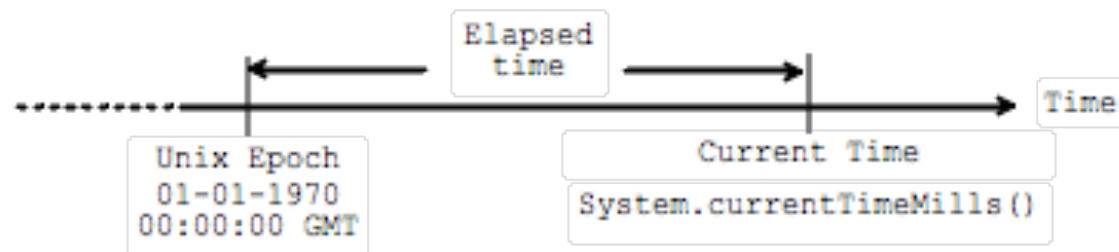$$celsius = (\tfrac{5}{9})(fahrenheit - 32)$$

Note: you have to write
celsius = (5.0 / 9) * (fahrenheit – 32)

Problem: Showing Current Time

Write a program that displays current time in GMT in the format hour:minute:second such as 1:45:19.

Using *currentTimeMillis()* method in *System* class returns the current time in milliseconds since the midnight, January 1, 1970 GMT.

ShowCurrentTime.java

Augmented Assignment Operators

Operator	Name	Example	Equivalent
+=	Addition assignment	i += 8	i = i + 8
-=	Subtraction assignment	i -= 8	i = i - 8
*=	Multiplication assignment	i *= 8	i = i * 8
/=	Division assignment	i /= 8	i = i / 8
%=	Remainder assignment	i %= 8	i = i % 8

Increment and Decrement Operators

Operator	Name	Description	Example (assume i = 1)
++var	preincrement	Increment var by 1, and use the new var value in the statement	`int j = ++i;` `// j is 2, i is 2`
var++	postincrement	Increment var by 1, but use the original var value in the statement	`int j = i++;` `// j is 1, i is 2`
--var	predecrement	Decrement var by 1, and use the new var value in the statement	`int j = --i;` `// j is 0, i is 0`
var--	postdecrement	Decrement var by 1, and use the original var value in the statement	`int j = i--;` `// j is 1, i is 0`

Increment and
Decrement Operators, cont.

```
int i = 10;
int newNum = 10 * i++;
```
Same effect as →
```
int newNum = 10 * i;
i = i + 1;
```

```
int i = 10;
int newNum = 10 * (++i);
```
Same effect as →
```
i = i + 1;
int newNum = 10 * i;
```

Increment and
Decrement Operators, cont.

Using increment and decrement operators makes expressions short, but it also makes them complex and difficult to read.

int k = ++i + i.

Assignment Expressions and Assignment Statements

Prior to Java 2, all the expressions can be used as statements.

Since Java 2, only the following types of expressions can be statements:

variable op= expression; // Where op is +, -, *, /, or %

++variable;

variable++;

--variable;

variable--;

Numeric Type Conversion

Consider the following statements:

```
byte i = 100;
long k = i * 3 + 4;
double d = i * 3.1 + k / 2;
```

Conversion Rules

When performing a binary operation involving two operands of different types, Java automatically converts the operand based on the following rules:

1. If one of the operands is double, the other is converted into double.
2. Otherwise, if one of the operands is float, the other is converted into float.
3. Otherwise, if one of the operands is long, the other is converted into long.
4. An integer literal can be assigned to an integer variable as long as it can fit into the variable.
5. Otherwise, both operands are converted into int.

Type Casting

Implicit casting
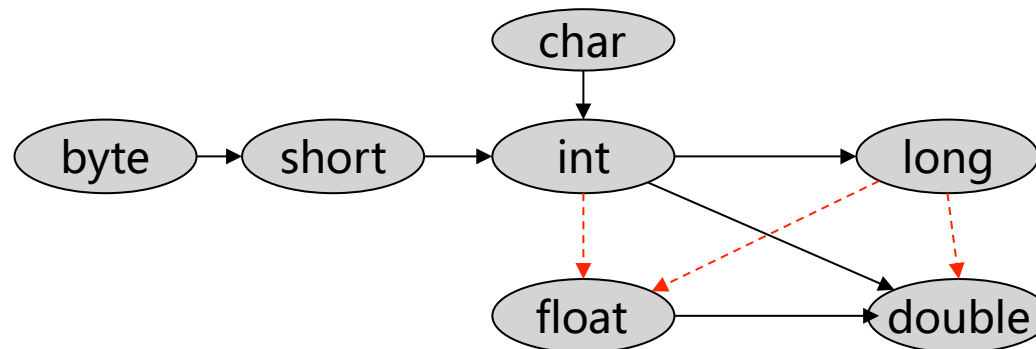   ***double d = 3;***   (type widening)

Explicit casting
   int i = (int)3.0;   (type narrowing)
   int i = (int)3.9;   (Fraction part is truncated)

What is wrong?   *int x = 5 / 2.0;*

Narrowing -- --->Widening
    byte, short, char---> int--> long--> float--> double

Casting in an Augmented Expression

An augmented expression of the form x1 op= x2 is implemented as x1 = (T)(x1 op x2), where T is the type for x1.

Therefore

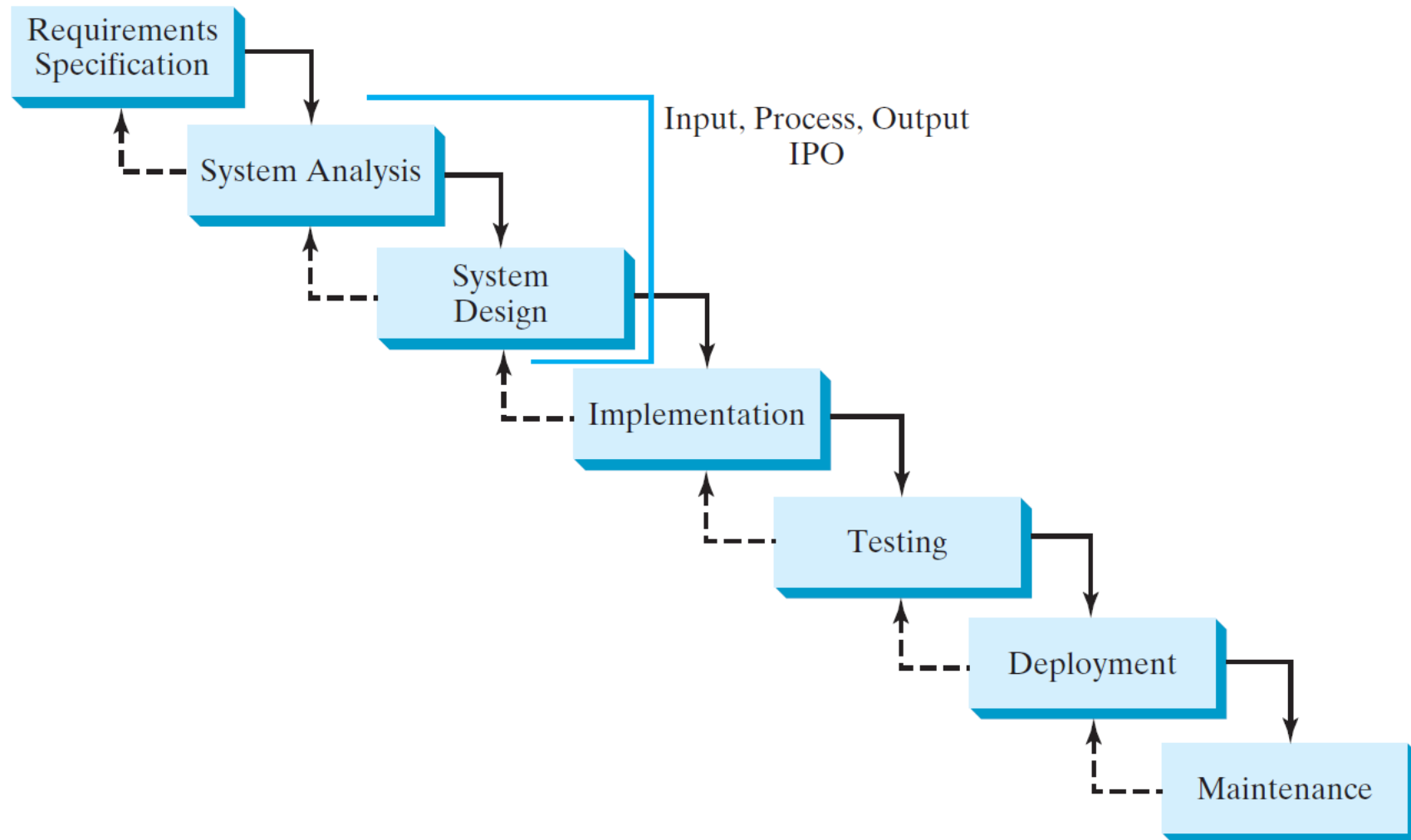> *int sum = 0;*
> *sum += 4.5;*

is correct

> *sum += 4.5* is equivalent to *sum = (int)(sum + 4.5)*
> *// sum becomes 4 after this statement*

Boolean type and operators

- *boolean b=**false**;*

- *true and false are literals*
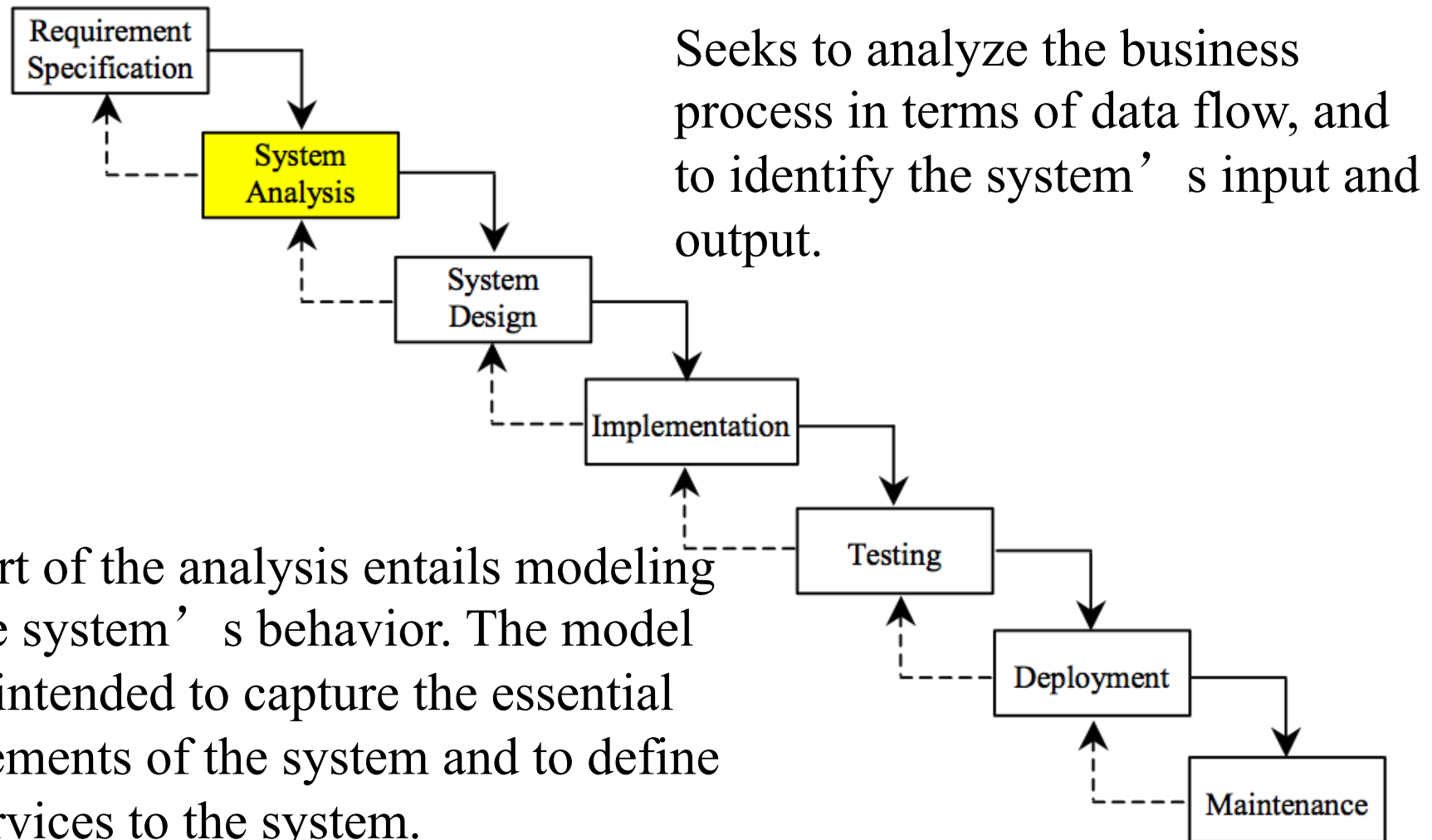
Software Development Process *

Requirements Specification → System Analysis → System Design → Implementation → Testing → Deployment → Maintenance

Input, Process, Output
IPO

Requirement Specification *

Requirement
Specification

System
Analysis

System
Design

Implementation

Testing

Deployment

Maintenance

A formal process that seeks to understand the problem and document in detail what the software system needs to do. This phase involves close interaction between users and designers.
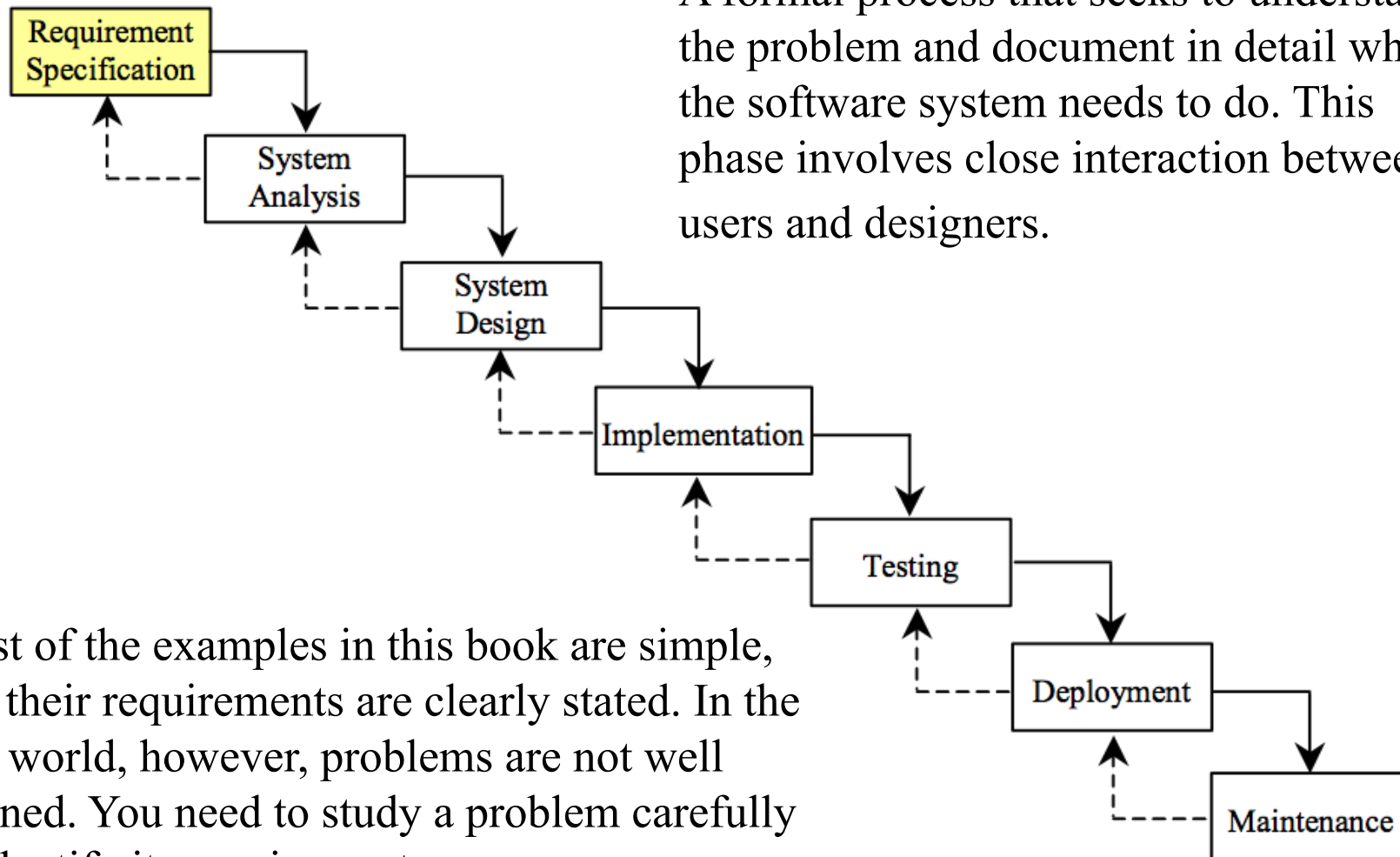
Most of the examples in this book are simple, and their requirements are clearly stated. In the real world, however, problems are not well defined. You need to study a problem carefully to identify its requirements.
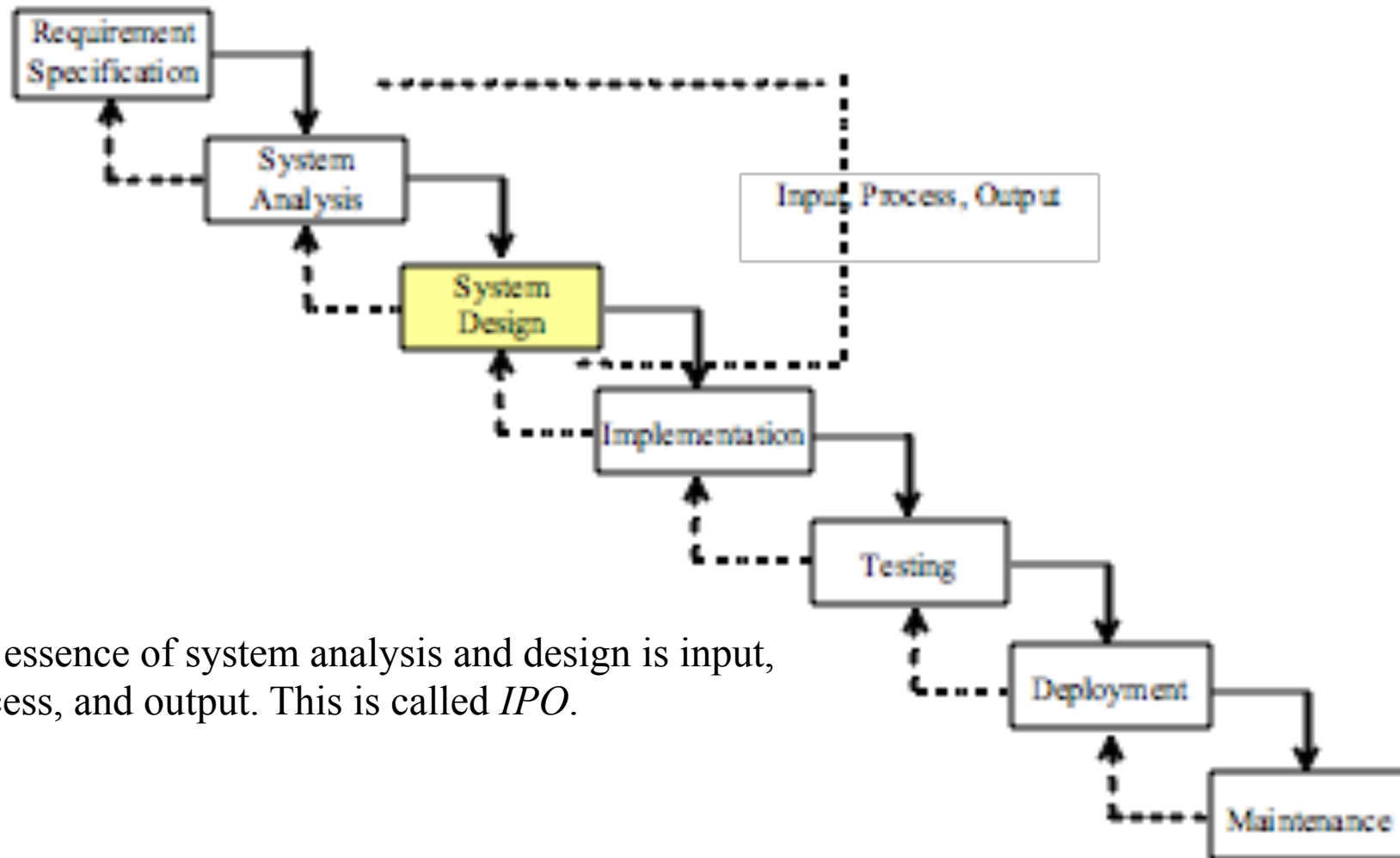
System Analysis *

Requirement Specification → System Analysis → System Design → Implementation → Testing → Deployment → Maintenance

Seeks to analyze the business process in terms of data flow, and to identify the system's input and output.

Part of the analysis entails modeling the system's behavior. The model is intended to capture the essential elements of the system and to define services to the system.

50

System Design *

Requirement Specification → System Analysis → System Design → Implementation → Testing → Deployment → Maintenance

The process of designing the system's components.
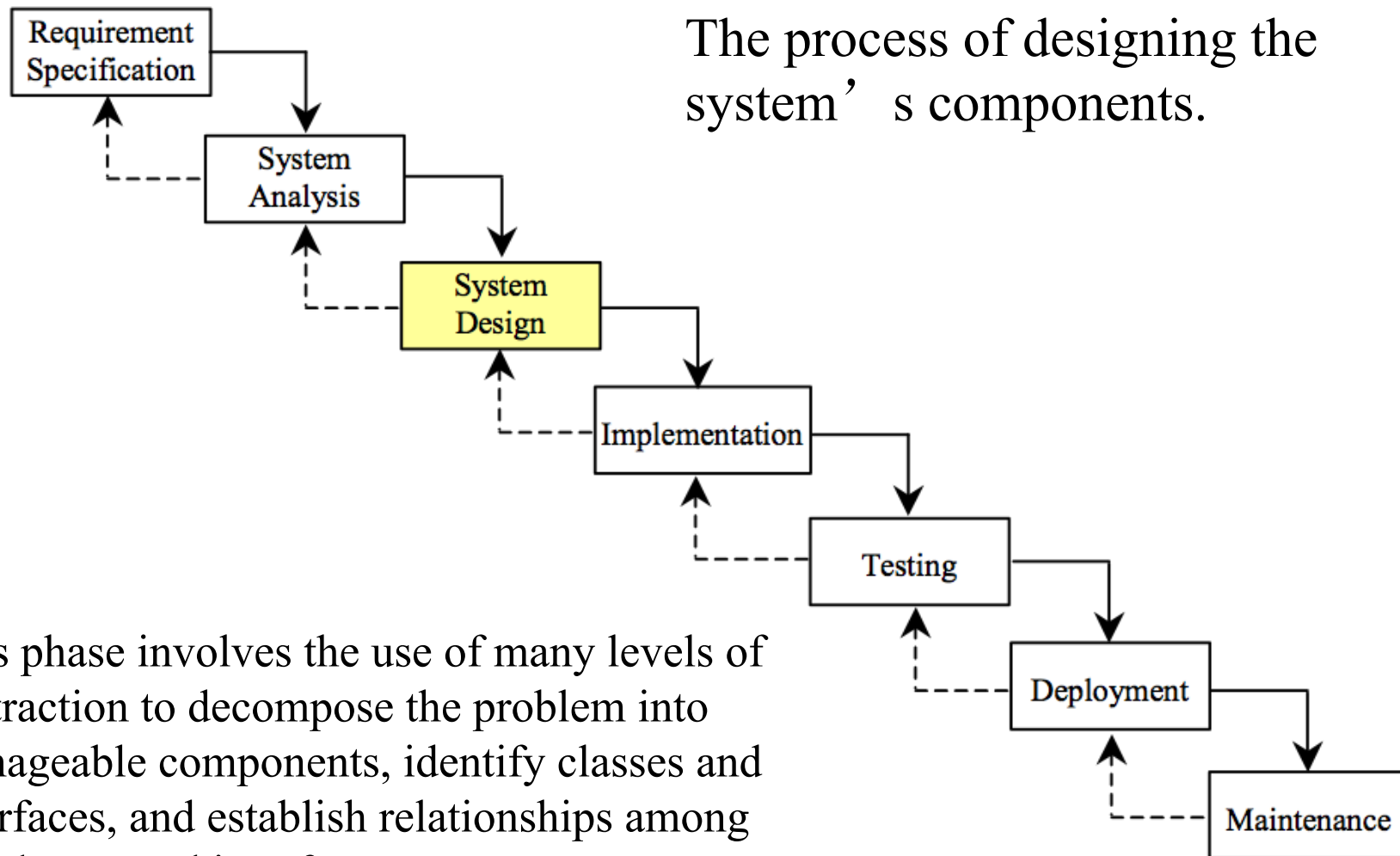
This phase involves the use of many levels of abstraction to decompose the problem into manageable components, identify classes and interfaces, and establish relationships among the classes and interfaces.
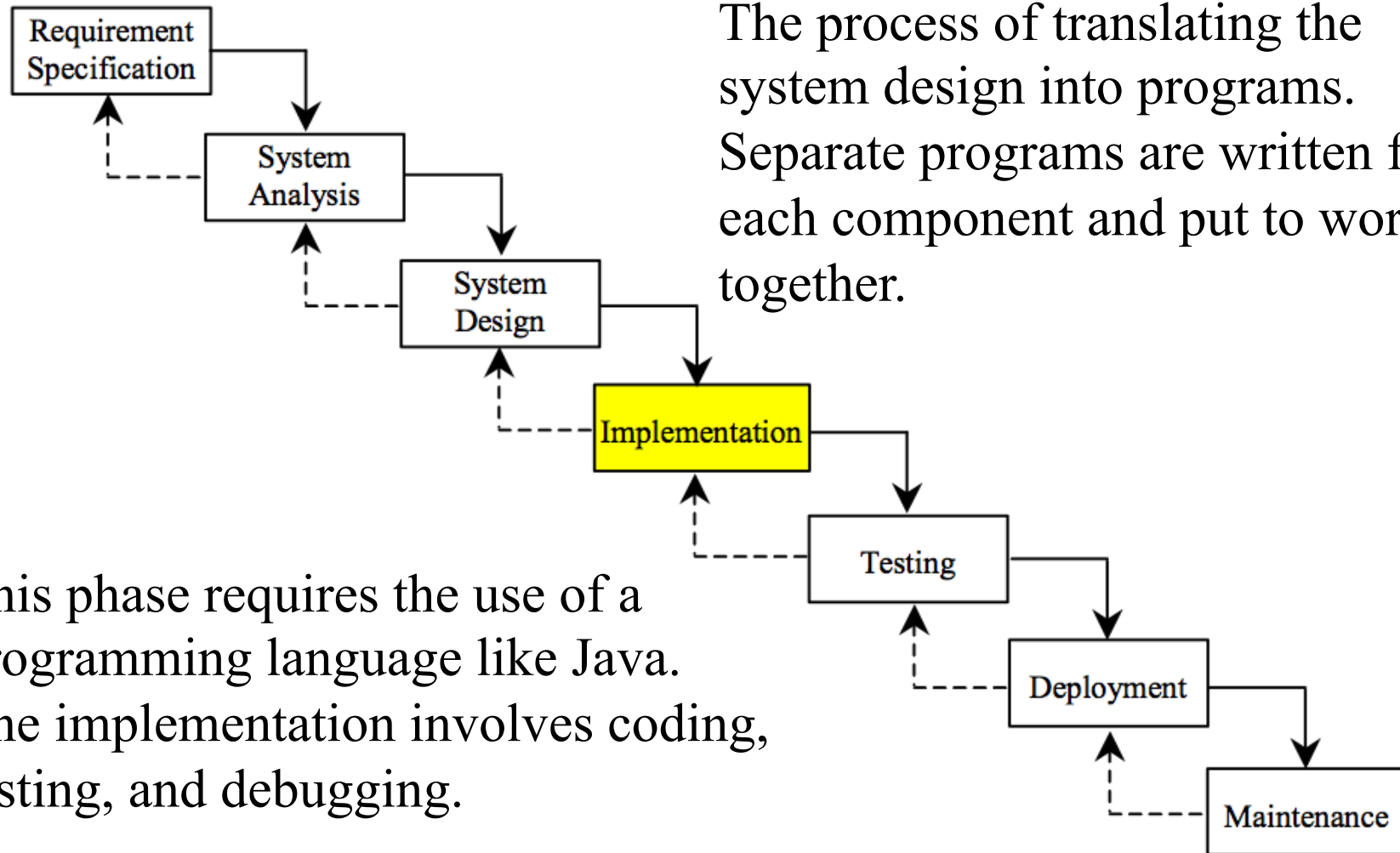
IPO *

The essence of system analysis and design is input, process, and output. This is called *IPO*.
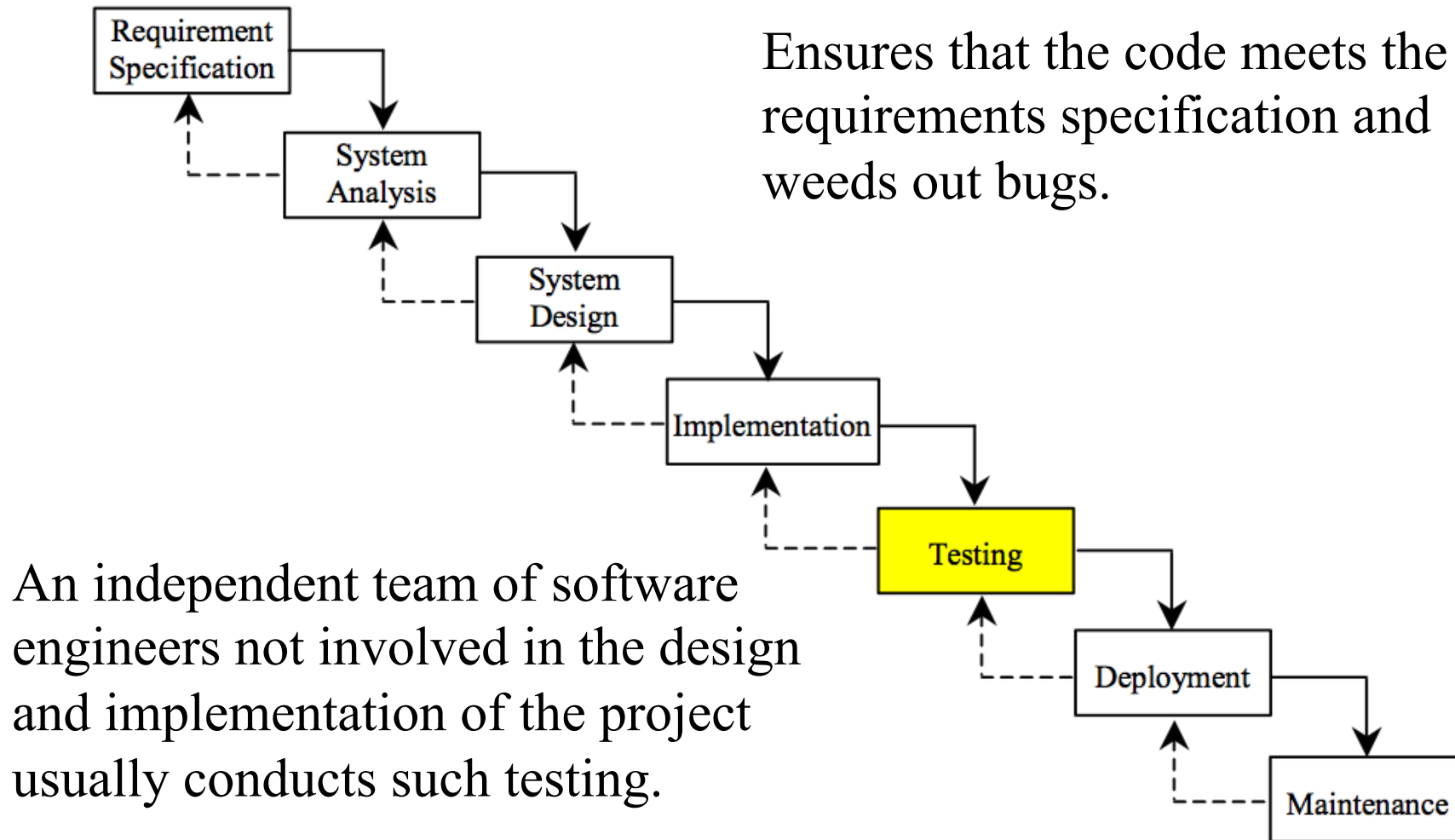
Implementation *

The process of translating the system design into programs. Separate programs are written for each component and put to work together.

This phase requires the use of a programming language like Java. The implementation involves coding, testing, and debugging.
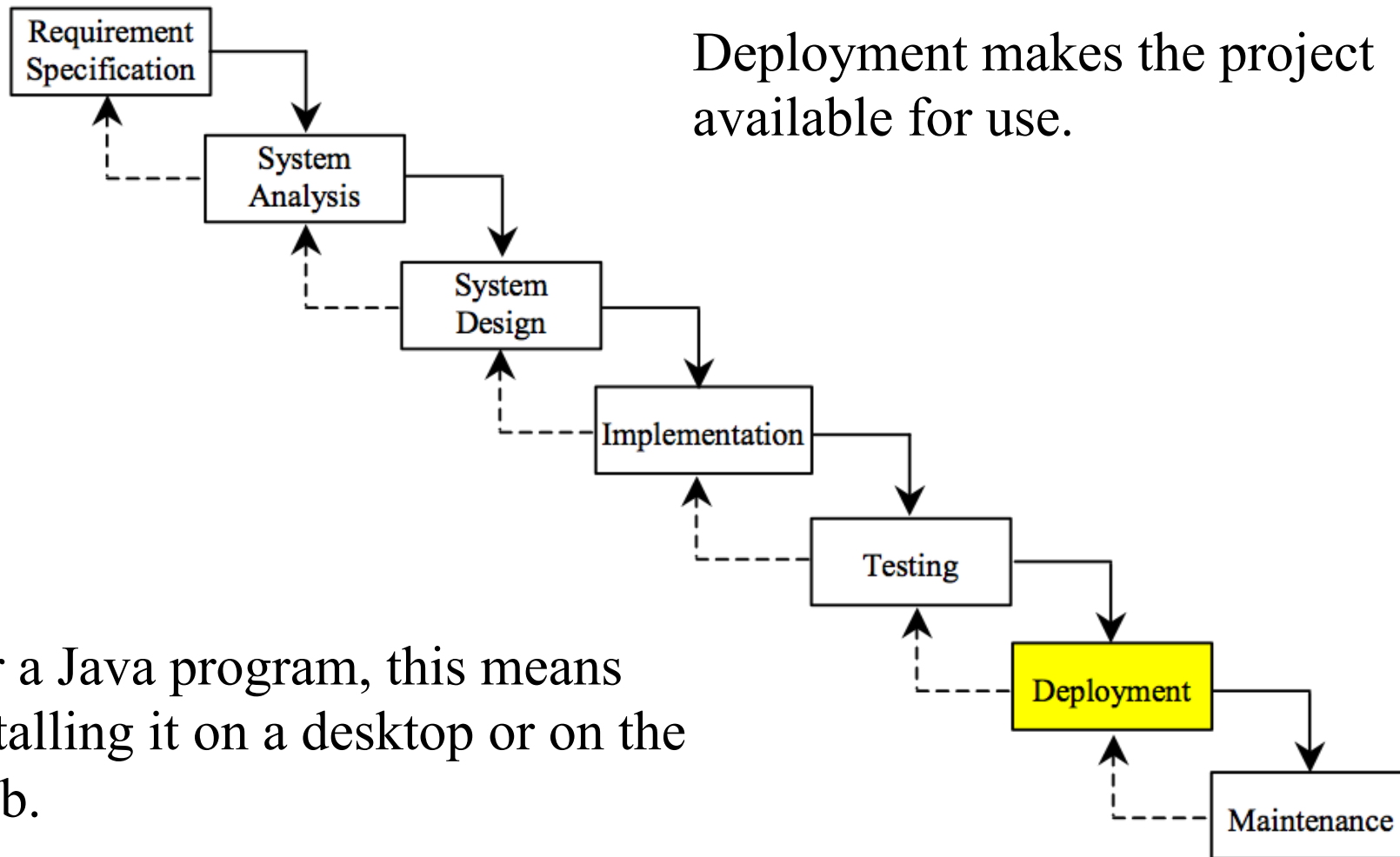
Testing *

Ensures that the code meets the requirements specification and weeds out bugs.

An independent team of software engineers not involved in the design and implementation of the project usually conducts such testing.
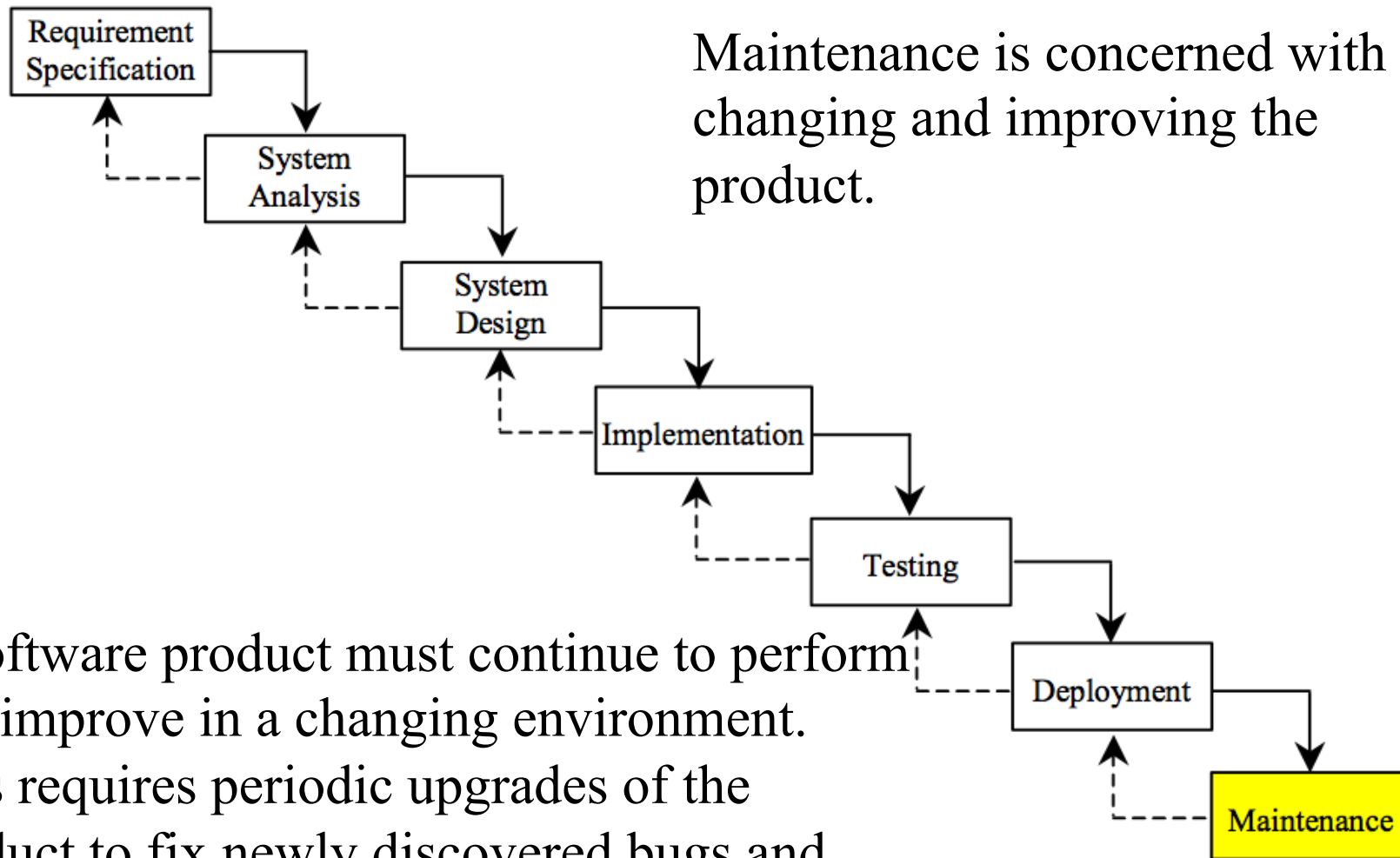
Deployment *

Deployment makes the project available for use.

For a Java program, this means installing it on a desktop or on the Web.

Maintenance *

Maintenance is concerned with changing and improving the product.

A software product must continue to perform and improve in a changing environment. This requires periodic upgrades of the product to fix newly discovered bugs and incorporate changes.

56

Problem:
Computing Loan Payments

This program lets the user enter the interest rate, number of years, and loan amount, and computes monthly payment and total payment.

$$monthlyPayment = \frac{loanAmount \times monthlyInterestRate}{1 - \dfrac{1}{(1 + monthlyInterestRate)^{numberOfYears \times 12}}}$$

Problem: Monetary Units

This program lets the user enter the amount in decimal representing dollars and cents and output a report listing the monetary equivalent in single dollars, quarters, dimes, nickels, and pennies.

Supplement reading:
Common Errors and Pitfalls

✦ Common Error 1: Undeclared/Uninitialized Variables and Unused Variables

✦ Common Error 2: Integer Overflow

✦ Common Error 3: Round-off Errors

✦ Common Error 4: Unintended Integer Division

✦ Common Error 5: Redundant Input Objects

✦ Common Pitfall 1: Redundant Input Objects

Common Error 1: Undeclared/Uninitialized Variables and Unused Variables

double *interestRate =* *0.05*;

double *interest = interestrate * 45*;

Common Error 2: Integer Overflow

int *value = **2147483647** + **1**;*

// value will actually be -2147483648

Common Error 3: Round-off Errors

*System.out.println(**1.0 - 0.1 - 0.1 - 0.1 - 0.1 - 0.1**);*

*System.out.println(**1.0 - 0.9**);*

Common Error 4: Unintended Integer Division

int number1 = 1;

int number2 = 2;

double average=(number1+number2) / 2;

double average=(number1+number2) / 2.0;

Common Pitfall 1: Redundant Input Objects

*Scanner input = **new** Scanner(System.in);*

*System.out.print(**"Enter an integer: "**);*

int *v1 = input.nextInt();*

*Scanner input1 = **new** Scanner(System.in);*

*System.out.print(**"Enter a double value: "**);*

double *v2 = input1.nextDouble();*

Exercise 01

1.1 包的使用：示例**DroidApp**。拷贝**./ch01/huz/**目录，构建带子目录的文件夹，了解其包含的子目录结构，并尝试用命令行方式进行带包编译和带包执行，从而掌握目录、包、**classpath**等概念的关系。

1.2 2*2线性方程组求解可以使用**Cramer**规则

$$ax + by = e$$
$$cx + dy = f$$

$$x = \frac{ed - bf}{ad - bc}$$

$$y = \frac{af - ec}{ad - bc}$$

编写程序，求解一下方程组并显示**x**和**y**的值。

$$3.4x + 50.2y = 44.5$$
$$2.1x + 0.55y = 5.9$$

Submission Deadline: the Next Monday(10/15/2018 9:00am(class03) or 11:10am(class04))

Exercise 01

1.3   财务应用：计算未来投资值
编写程序，读取投资总额、年利率和年数，然后使用下面的公式来显示未来投资金额：
未来投资金额=投资总额× （1+月利率）$^{\text{年数×12}}$
例如：如果输入的投资金额为**1000**，年利率为**3.25%**，年数为**1**，那么未来投资额为**1032.98**。

1.4   求出年数
编写程序，提示用户输入分钟数（例如十亿），然后显示这些分钟代表多少年和多少天。
为了简化问题，假设一年有**365天**。

1.5   选做题，完善
https://github.com/huzhengatUCSD/Java_Course/blob/master/ch02/Calculator.java的
未写好的方法，并试着编写一个
TestCalculator类，运用**Calculator**的方法完成对应运算功能。

```
//Submission Deadline: the Next Monday(10/16/2017
  9:00am(class03) or 11:10am(class04) )
```