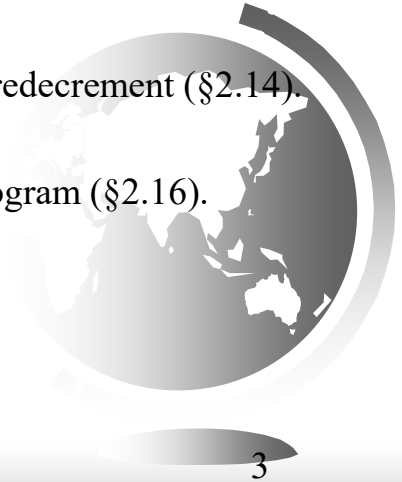
02 Elementary Programming

Motivations

In the preceding chapter, you learned how to create, compile, and run a Java program. Starting from this chapter, you will learn how to solve practical problems programmatically. Through these problems, you will learn Java primitive data types and related subjects, such as variables, constants, data types, operators, expressions, and input and output.

Objectives

- To write Java programs to perform simple computations (§2.2).
- To obtain input from the console using the **Scanner** class (§2.3).
- To use identifiers to name variables, constants, methods, and classes (§2.4).
- To use variables to store data (§§2.5–2.6).
- To program with assignment statements and assignment expressions (§2.6).
- To use constants to store permanent data (§2.7).
- To name classes, methods, variables, and constants by following their naming conventions (§2.8).
- To explore Java numeric primitive data types: **byte**, **short**, **int**, **long**, **float**, and **double** (§2.9.1).
- To read a **byte**, **short**, **int**, **long**, **float**, or **double** value from the keyboard (§2.9.2).
- To perform operations using operators +, **-**, *****, /, and **%** (§2.9.3).
- To perform exponent operations using **Math.pow(a, b)** (§2.9.4).
- To write integer literals, floating-point literals, and literals in scientific notation (§2.10).
- To write and evaluate numeric expressions (§2.11).
- To obtain the current system time using **System.currentTimeMillis()** (§2.12).
- To use augmented assignment operators (§2.13).
- To distinguish between postincrement and preincrement and between postdecrement and predecrement (§2.14).
- To cast the value of one type to another type (§2.15).
- To describe the software development process and apply it to develop the loan payment program (§2.16).
- To write a program that converts a large amount of money into smaller units (§2.17).
- To avoid common errors and pitfalls in elementary programming (§2.18).



Introducing Programming with an Example

Listing 2.1 Computing the Area of a Circle

This program computes the area of the circle.

ComputeArea

Animation

Run

animation

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);
  }
}
```

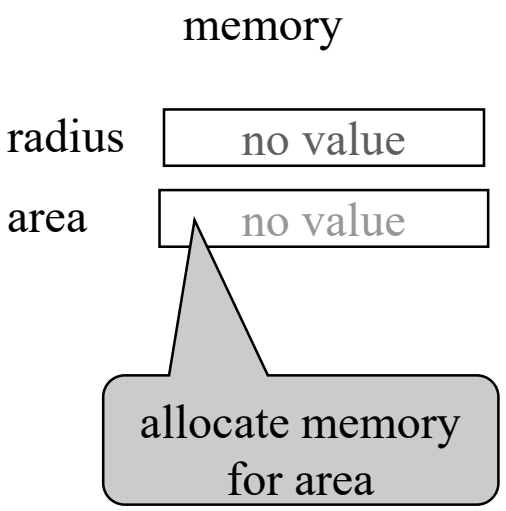
allocate memory for radius

radius | no value

animation

Trace a Program Execution

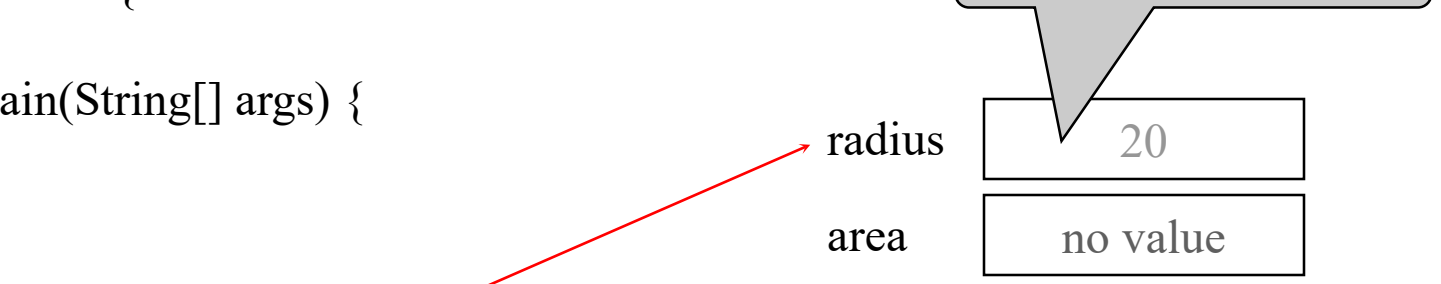
```
public class ComputeArea {
  /** Main method */
  public static void main(String[] args) {
    double radius;
    double area;

    // 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);
  }
}
```



animation

Trace a Program Execution

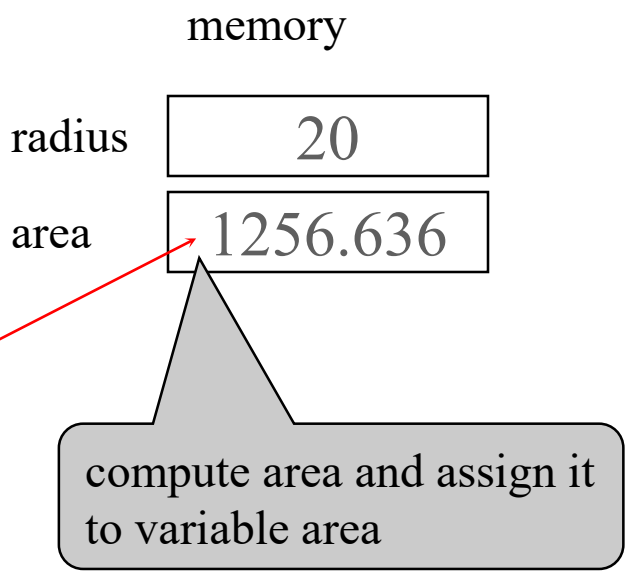
```
public class ComputeArea {
  /** Main method */
  public static void main(String[] args) {
    double radius;
    double area;

    // 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);
  }
}
```



animation

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);
  }
}
```



animation

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();
```

Animation

ComputeAreaWithConsoleInput

Run

ComputeAverage

Run

```java
import java.util.Scanner; // Scanner is in the java.util package

public class ComputeAreaWithConsoleInput {
  public static void main(String[] args) {
    // Create a Scanner object
    Scanner input = new Scanner(System.in);

    // Prompt the user to enter a radius
    System.out.print("Enter a number for radius: ");
    double radius = input.nextDouble();

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

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

Identifiers

- An identifier is a sequence of characters that consist of letters, digits, underscores (_), and dollar signs ($).
- An identifier must start with a letter, an underscore (_), or a dollar sign ($). It cannot start with a digit.
- An identifier cannot be a reserved word. (See Appendix A, "Java Keywords," for a list of reserved words).
- An identifier cannot be `true`, `false`, or `null`.
- An identifier can be of any length.
- 区分大小写

Which one below is NOT a valid Java identifier?

A. Int B. goto C. 变量 D. $0

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;
```

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;`

与C, C++的区别

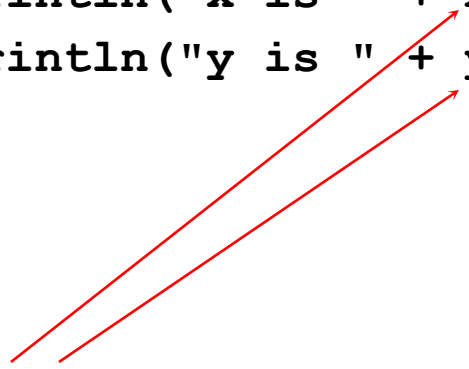
- C, C++区分变量的声明与定义
- int i = 10;是个定义，而
- external int i; 则是一个声明。

- 在java中，不区分变量的声明与定义

Java assigns no default value to a local variable inside a method.

C++中会怎么样？

```
public class Test {
  public static void main(String[] args) {
    int x; // x has no default value
    String y; // y has no default value
    System.out.println("x is " + x);
    System.out.println("y is " + y);
  }
}
```



Compile error: variable not initialized

- 在C/C++中，变量的初始化还是得依赖于程序员的自觉性。对于函数局部变量，编译器不会为基本类型赋予默认初始值，新手经常会使用未初始化的指针访问内存，导致程序崩溃。对于类对象，编译器将使用类的默认构造函数对对象进行初始化。
- 而在java中，对于方法的局部变量，java以编译时错误来保证变量在使用前都能得到恰当的初始化。

Given code below, which statement is correct?

```
public class Person{
     static int arr[] = new int[5];
     public static void main(String a[]) {
          System.out.println(arr[0]);
     }}
```

A. Compile error.

B. Compiles but run-time error

C. Prints 0

D. Prints nothing

Named Constants

```
final datatype CONSTANTNAME = VALUE;

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

Given a public member variable `MAX_LENGTH` as the `int` type is a constant of 100, the correct statement to define the variable is:

A. `public int MAX_LENGTH=100`
B. `final int MAX_LENGTH=100`
C. `public const int MAX_LENGTH=100`
D. `public final int MAX_LENGTH=100`

final VS. const

- final有三种主要用法：
  - 修饰变量:variable
    - final变量是不可改变的，但它的值可以在运行时刻初始化，也可以在编译时刻初始化，甚至可以放在构造函数中初始化，而不必在声明的时候初始化，所以下面的语句均合法：
      - final int i = 1; // 编译时刻
      - final int i2 = (int)(Math.Random() * 10); //运行时刻
      - final int i3; //构造函数里再初始化
    - 如果修饰类对象，并不表示这个对象不可更改，而是表示这个变量不可再赋成其它对象，这就比较象 C++的 Class const * p
      - final Value v = new Value(); v = new Value(); //不允许！

final VS. const

- final有三种主要用法：

(1)const修饰成员变量

const修饰类的成员函数，表示成员常量，不能被修改，同时它只能在初始化列表中赋值。

```
class A
{
    ...
    const int nValue;        //成员常量不能被修改
    ...
    A(int x): nValue(x) { } ; //只能在初始化列表中赋值
 }
```

C++中

  - 修饰方法:method
    - final表示一个Java函数不可更改，也就是不能被重载了，而不是修饰返回值的
  - 修饰类:class
    - 表示整个类不能被继承了， 自然，里面的所有方法也相当于被加了final。

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

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

- Java中，整型的范围与运行JAVA代码的机器无关，解决了平台移植的问题。
- C，C++中需要针对不同的处理器选择最为高效的整型，就造成了一个在32位处理器上运行很好的程序在16位系统上运行是发生整数溢出。

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

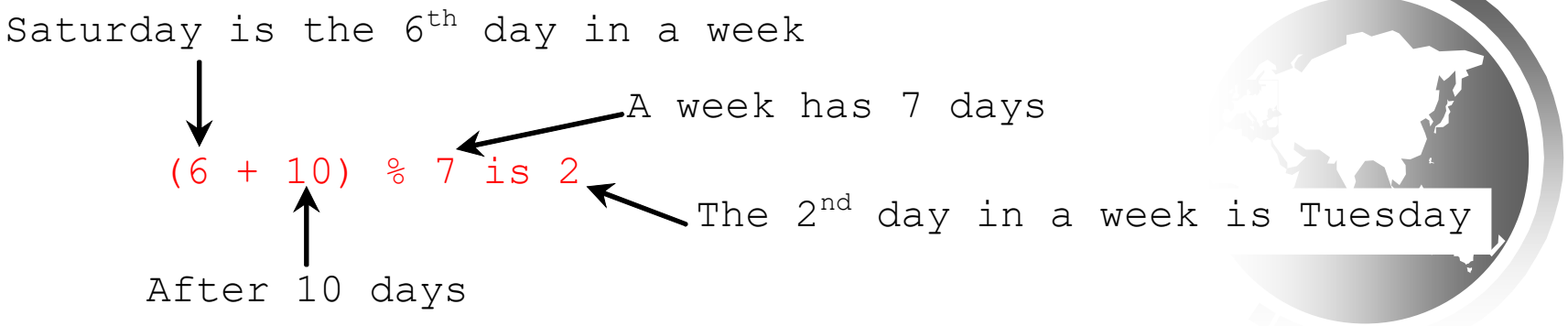
5 % 2 yields 1 (the remainder of the division)

Remainder Operator

Remainder is very useful in programming.

For example, an even number % 2 is always 0 and an odd number % 2 is always 1. So you can use this property to determine whether a number is even or odd.

Suppose today is Saturday and you and your friends are going to meet in 10 days. What day is in 10 days? You can find that day is Tuesday using the following expression:



Problem: Displaying Time

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

DisplayTime

Run

NOTE

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

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

displays 0.5000000000000001, not 0.5, and

System.out.println(1.0 - 0.9);

displays 0.09999999999999998, not 0.1. Integers are stored precisely. Therefore, calculations with integers yield a precise integer result.

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

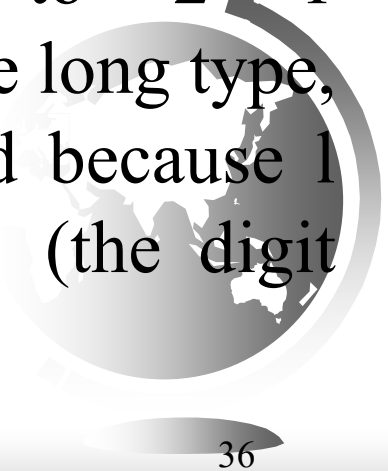
A *literal* is a constant value that appears directly in the program. For example, 34, 1,000,000, and 5.0 are literals in the following statements:

```
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. For example, the statement byte b = 1000 would cause a compilation error, because 1000 cannot be stored in a variable of the byte type.

An integer literal is assumed to be of the int type, whose value is between -2^{31} (-2147483648) to $2^{31}-1$ (2147483647). To denote an integer literal of the long type, append it with the letter L or l. L is preferred because l (lowercase L) can easily be confused with 1 (the digit one).



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. For example, 5.0 is considered a double value, not a float 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. For example, you can use 100.2f or 100.2F for a float number, and 100.2d or 100.2D for a double number.

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);
```

displays `1.0 / 3.0 is 0.3333333333333333`

16 digits

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

displays `1.0F / 3.0F is 0.33333334`

7 digits

特殊的浮点数值

- 正无穷大：Double.POSITIVE_INFINITY
- 负无穷大：Double.NEGATIVE_INFINITY
- NaN（不是一个数字）: Double.NaN

- 用 Double.isNaN来判断是否是数字

Scientific Notation

Floating-point literals can also be specified in scientific notation, for example, 1.23456e+2, same as 1.23456e2, is equivalent to 123.456, and 1.23456e-2 is equivalent to 0.0123456. E (or e) represents an exponent and it can be either in lowercase or uppercase.

Arithmetic Expressions

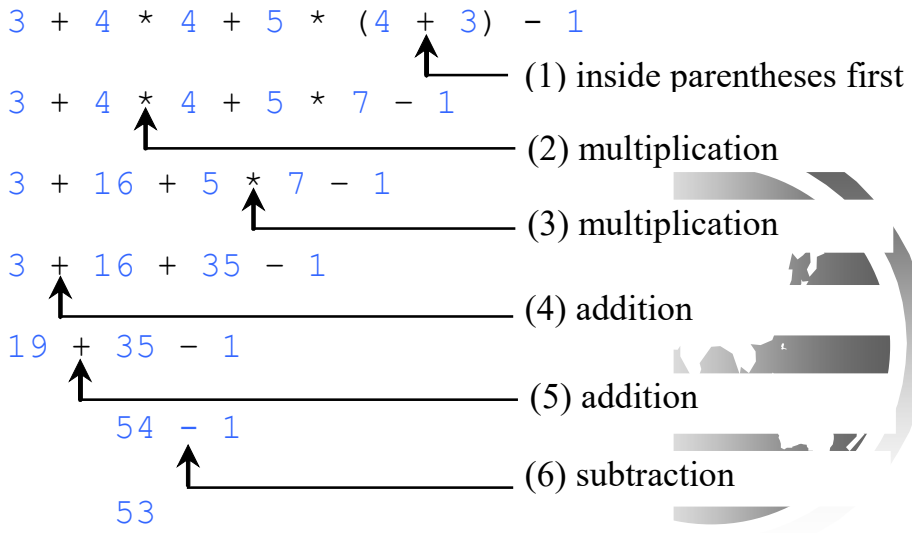
$$\frac{3+4x}{5} - \frac{10(y-5)(a+b+c)}{x} + 9\left(\frac{4}{x} + \frac{9+x}{y}\right)$$

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

Though Java has its own way to evaluate an expression behind the scene, the result of a Java expression and its corresponding arithmetic expression are the same. Therefore, you can safely apply the arithmetic rule for evaluating a Java expression.



Problem: Converting Temperatures

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

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

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

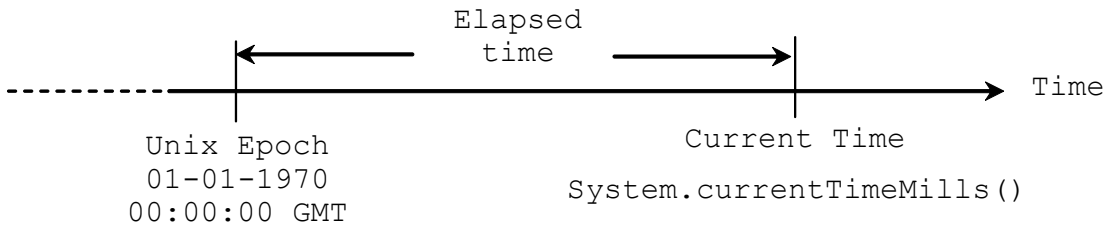
FahrenheitToCelsius

Run

Problem: Displaying Current Time

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

The **currentTimeMillis** method in the System class returns the current time in milliseconds since the midnight, January 1, 1970 GMT. (1970 was the year when the Unix operating system was formally introduced.) You can use this method to obtain the current time, and then compute the current second, minute, and hour as follows.



```java
public class ShowCurrentTime {
  public static void main(String[] args) {
    // Obtain the total milliseconds since midnight, Jan 1, 1970
    long totalMilliseconds = System.currentTimeMillis();

    // Obtain the total seconds since midnight, Jan 1, 1970
    long totalSeconds = totalMilliseconds / 1000;

    // Compute the current second in the minute in the hour
    long currentSecond = totalSeconds % 60;

    // Obtain the total minutes
    long totalMinutes = totalSeconds / 60;

    // Compute the current minute in the hour
    long currentMinute = totalMinutes % 60;

    // Obtain the total hours
    long totalHours = totalMinutes / 60;

    // Compute the current hour
    long currentHour = totalHours % 24;

    // Display results
    System.out.println("Current time is " + currentHour + ":"
      + currentMinute + ":" + currentSecond + " GMT");
  }
}
```

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++;
```

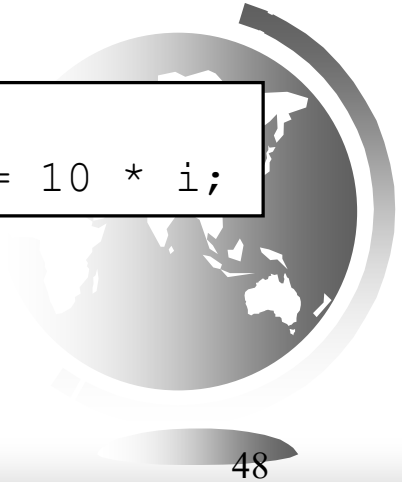
Same effect as

```
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.

Avoid using these operators in expressions that modify multiple variables, or the same variable for multiple times such as this: 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:

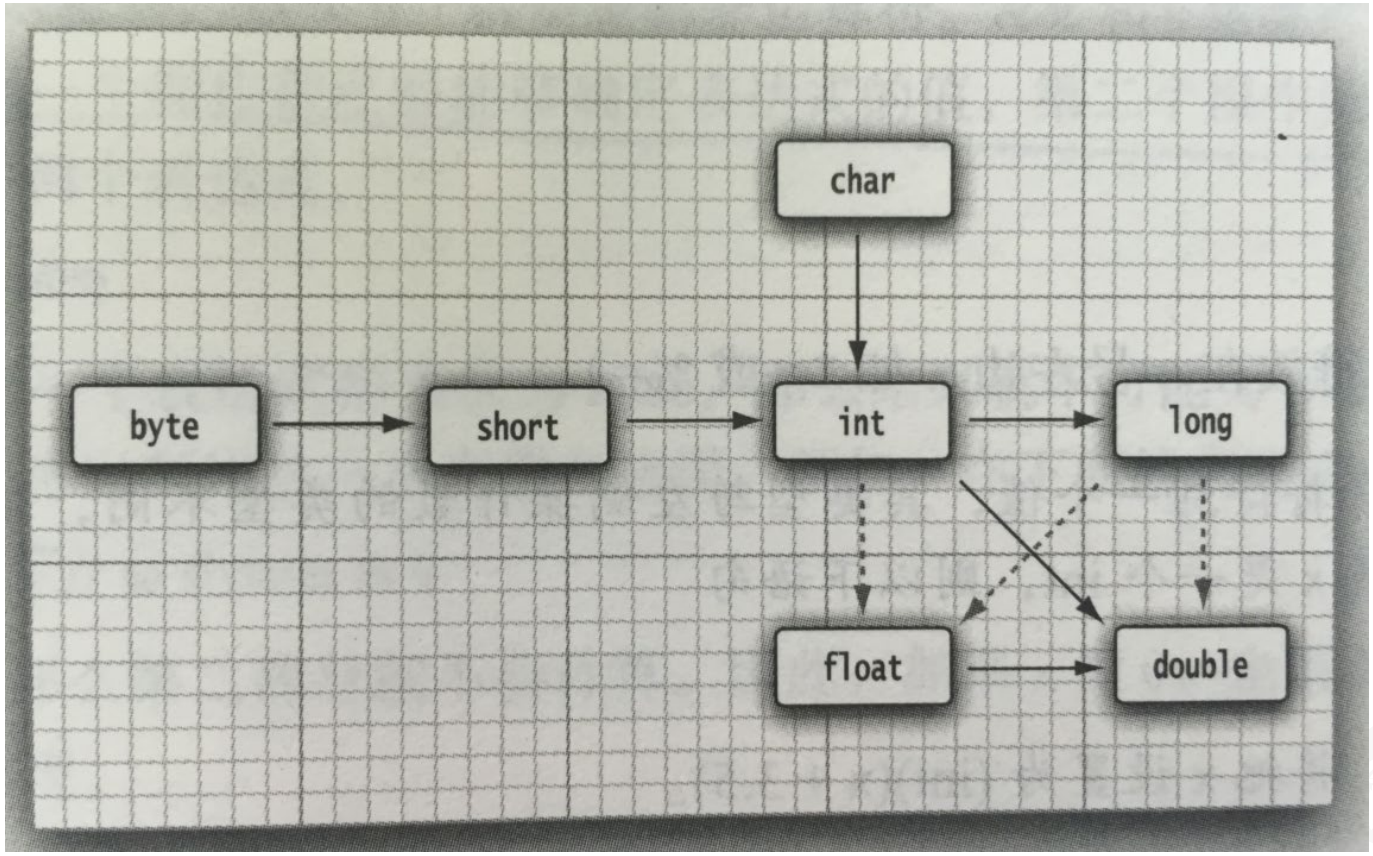
```
byte i = 100;
long k = i * 3 + 4;
double d = i * 3.1 + k / 2;
```

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. Otherwise, both operands are converted into int.

Type Casting



实线表示无信息丢失的转换；
虚线表示可能有精度损失的转换

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;

Type Casting

- double x = 9.997，求四舍五入的整数。
- int nx = (int)x;
- int nx = (int)Math.round(x);
- 为什么round后还需要强制转换？
- round返回结果是long类型。

Problem: Keeping Two Digits After Decimal Points

Write a program that displays the sales tax with two digits after the decimal point.

0.4445→0.44 ?

SalesTax

Run

```java
import java.util.Scanner;

public class SalesTax {
  public static void main(String[] args) {
    Scanner input = new Scanner(System.in);

    System.out.print("Enter purchase amount: ");
    double purchaseAmount = input.nextDouble();

    double tax = purchaseAmount * 0.06;
    System.out.println("Sales tax is " + (int)(tax * 100) / 100.0);
  }
}
```

Casting in an Augmented Expression

In Java, 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, the following code is correct.

```
int sum = 0;
sum += 4.5; // sum becomes 4 after this statement
```

sum += 4.5 is equivalent to **sum = (int)(sum + 4.5)**.

Java VS. C++

- 在c和c++中有时出现数据类型的隐含转换，这就涉及了自动强制类型转换问题。例如，在c++中可将一浮点值赋予整型变量，并去掉其尾数。**Java不支持c++中的自动强制类型转换，如果需要，必须由程序显式进行强制类型转换。**

Java是一个严格进行类型检查的程序语言，对于下面这样的程序，在C++的编译器上编译时最多只会出现警告的信息，但是在Java里则不予通过：

Int aInteger； Double aDouble=2.71828； AInteger = aDouble；

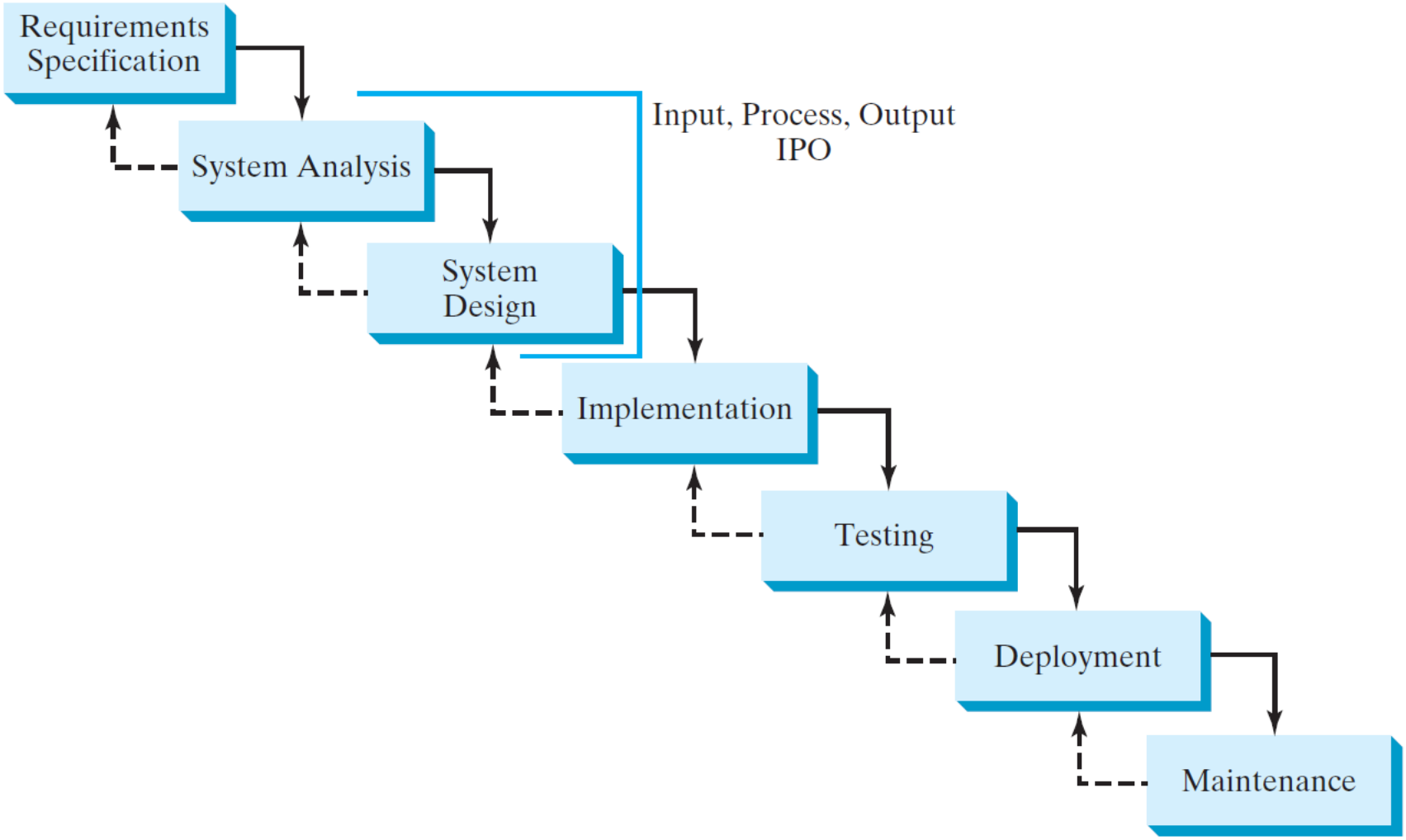
虽然这样的转型在C++里是合法的，但是也会造成数据精确度的损失。Java为了要确定写程序的人充分地了解这点，必须要程序设计强制转型(type casting)，Java的编译器才会接受：

int aInteger；

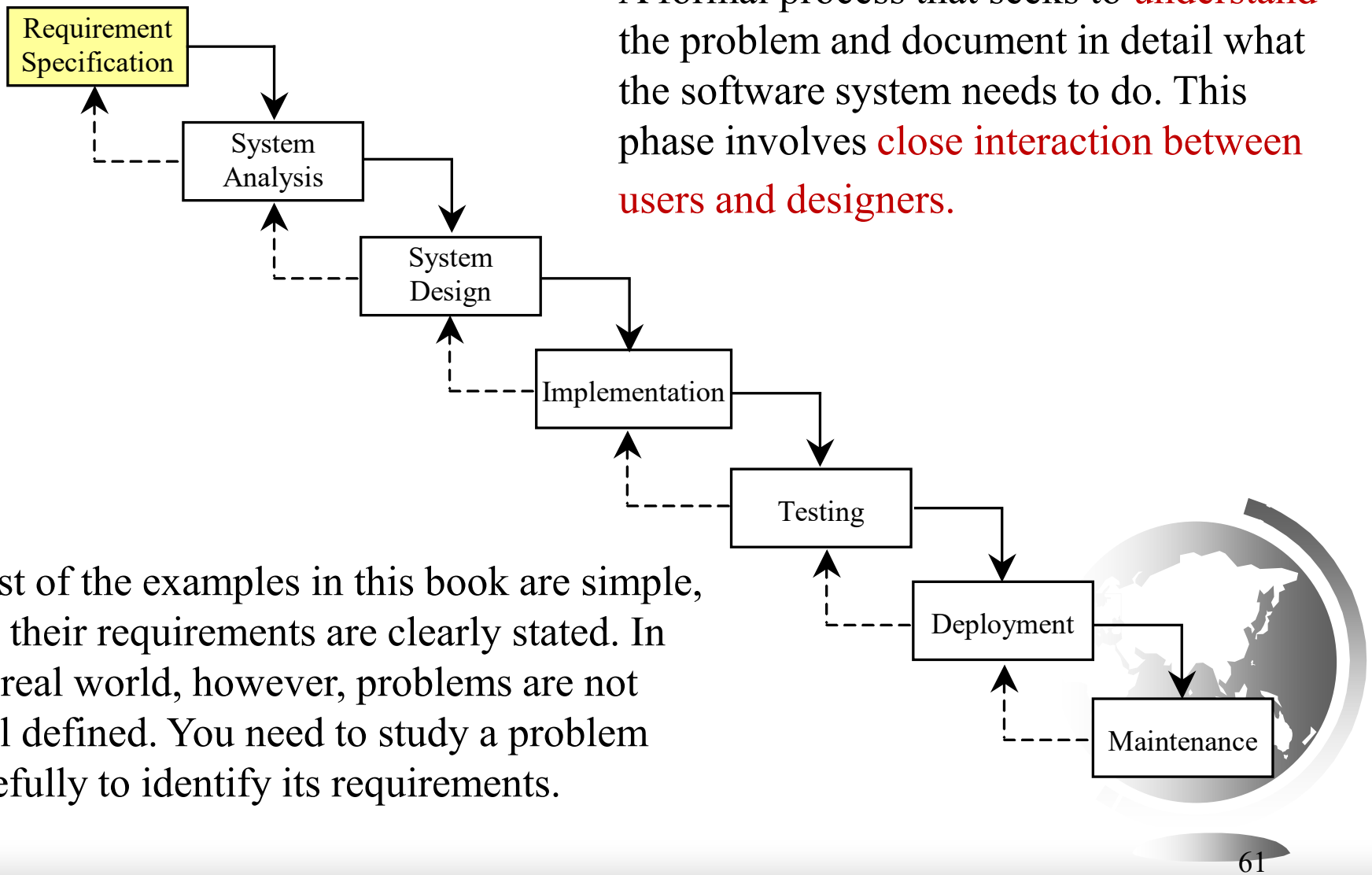
doublea Double=2.71828；

aInteger=(int)aDouble；

Software Development Process



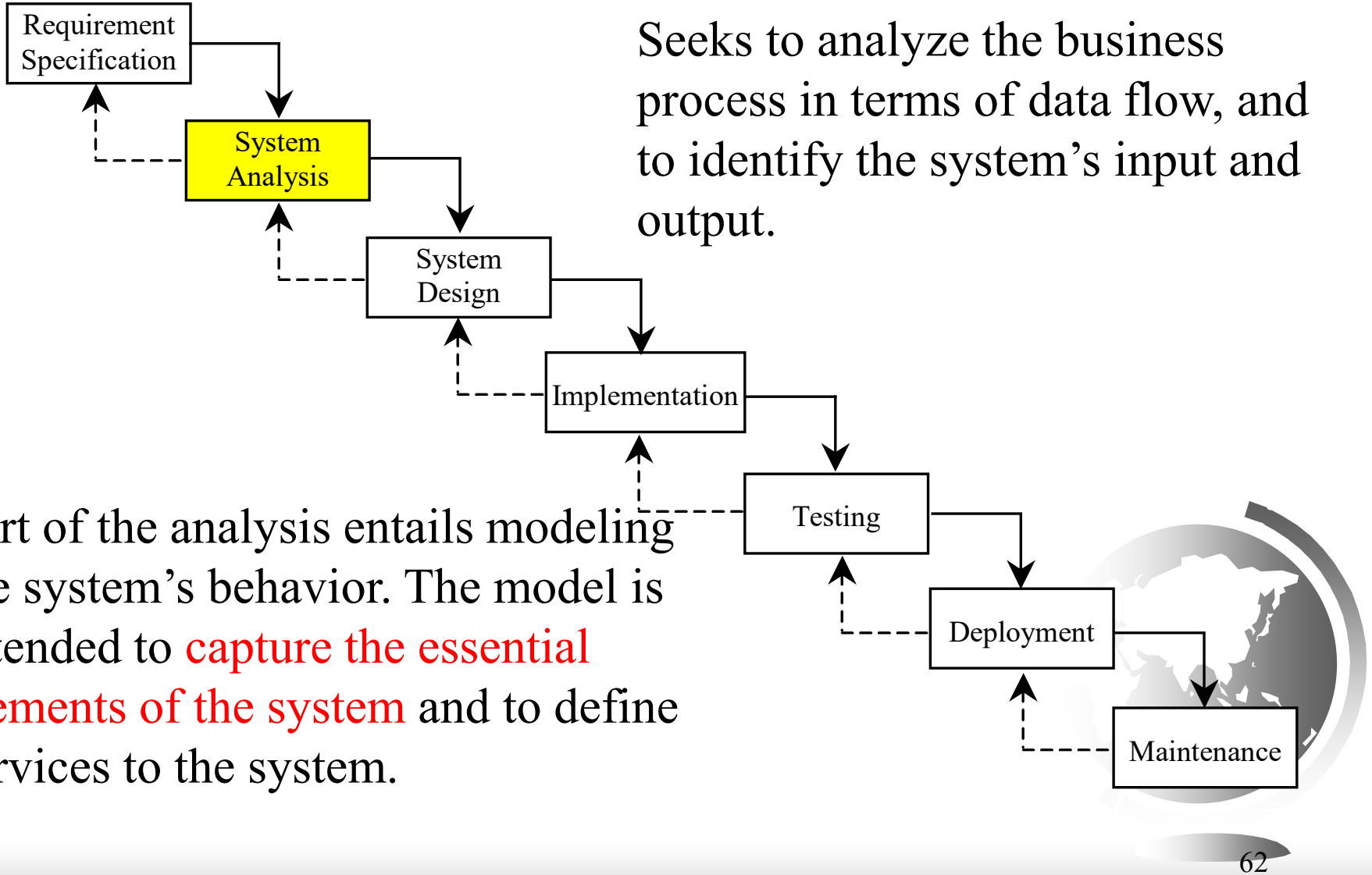
Requirement Specification



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.

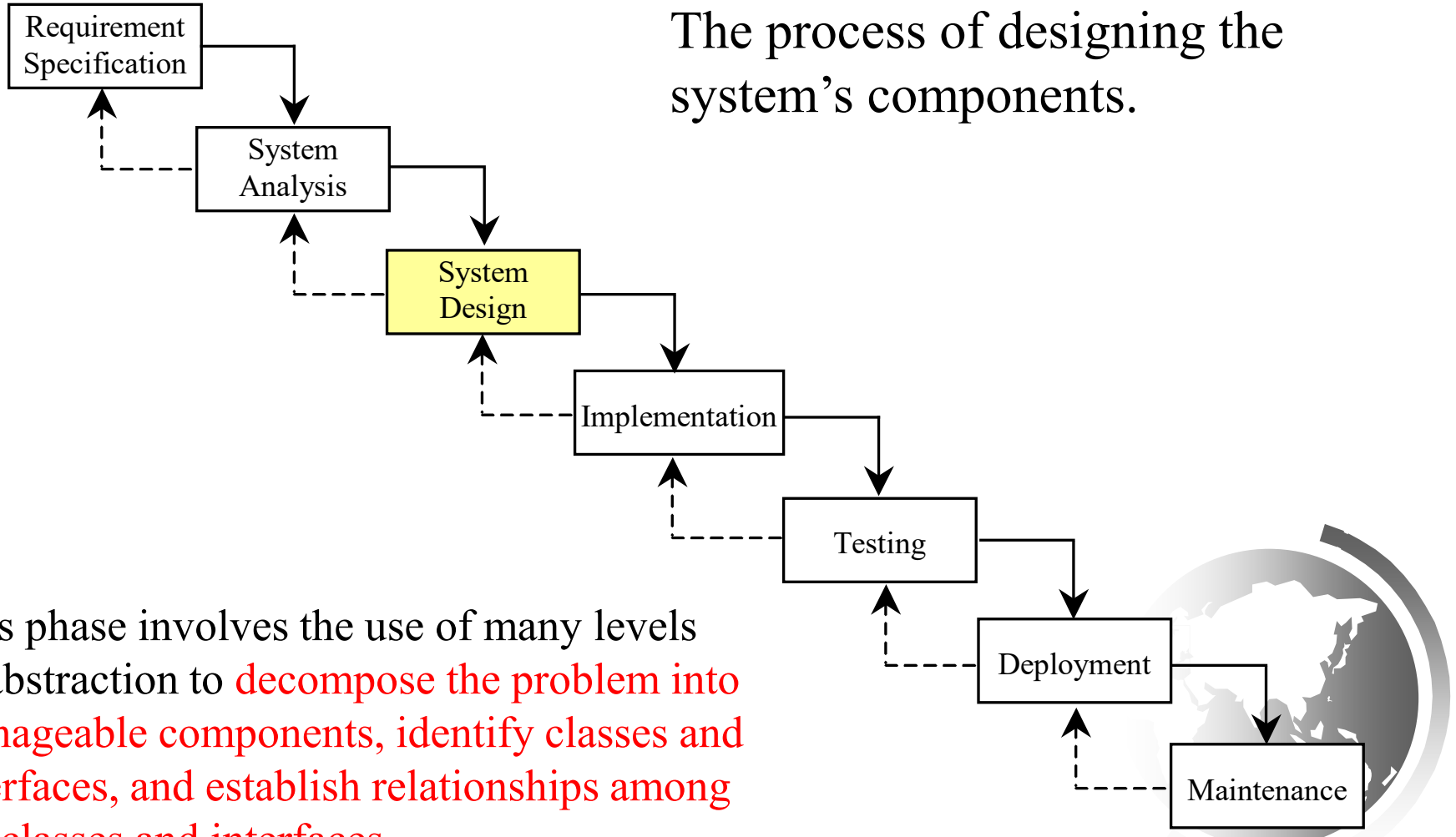
System Analysis



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.

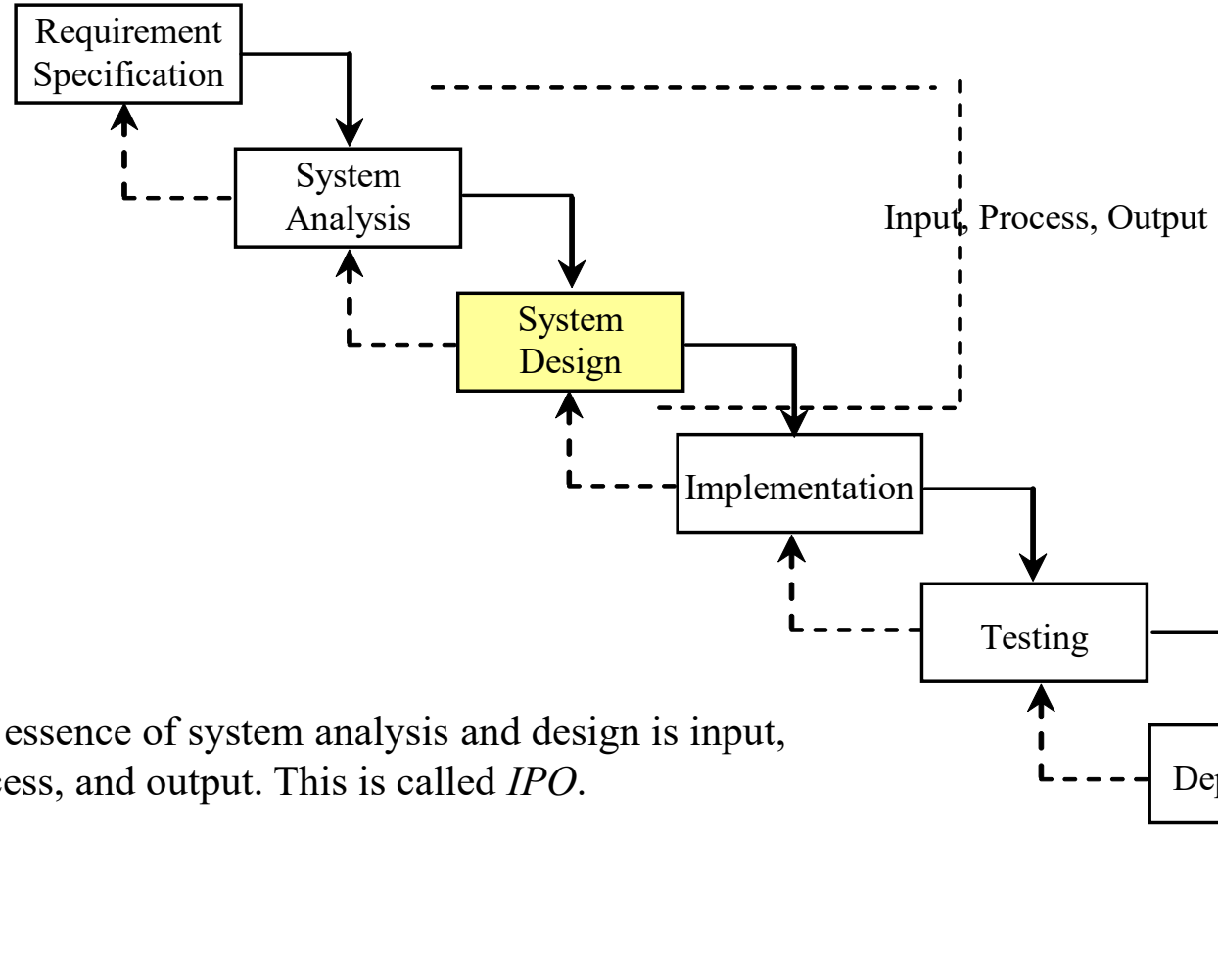
System Design



The process of designing the system's components.

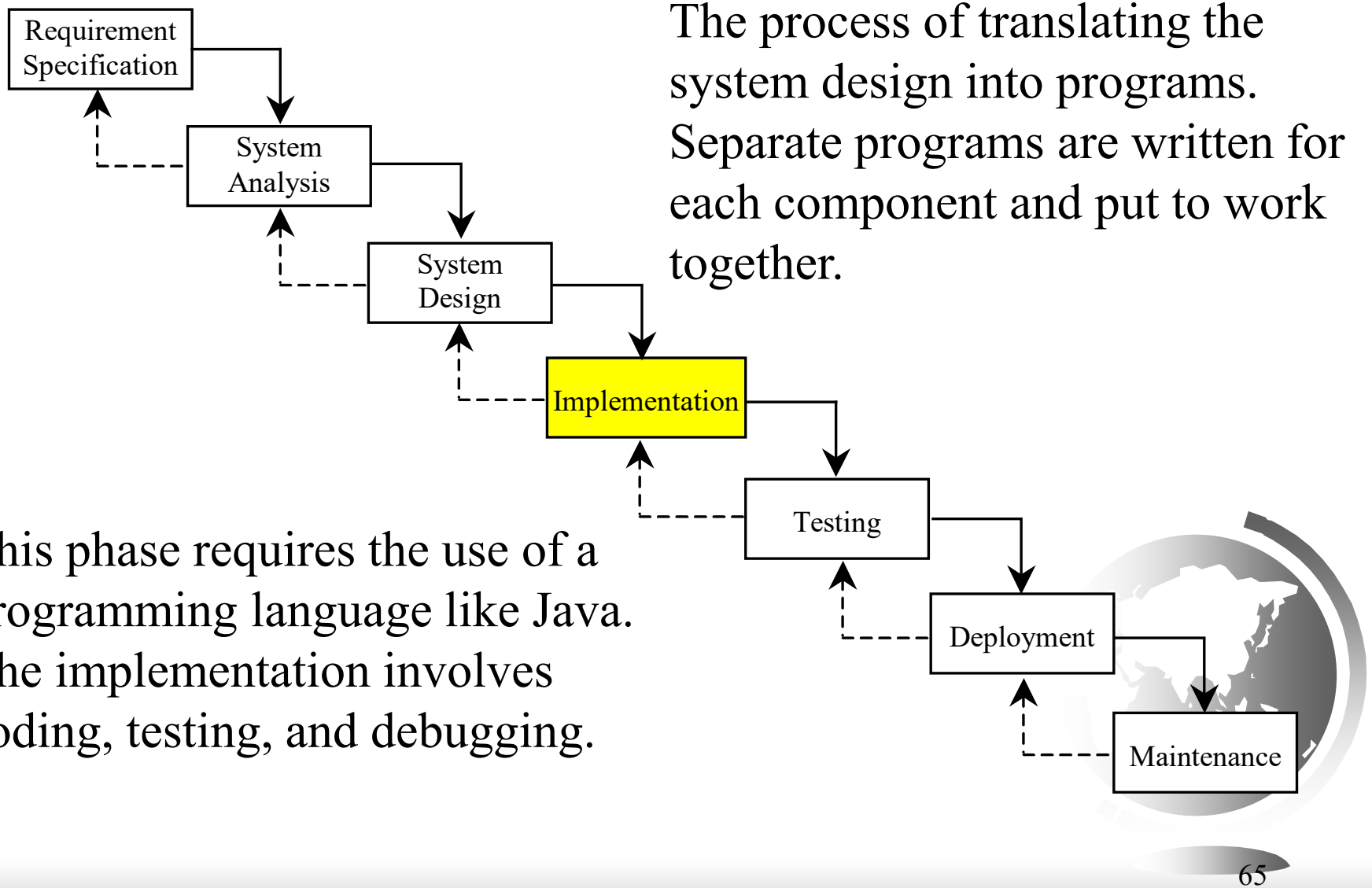
This phase involves the use of many levels of abstraction to decompose the problem into manageable components, identify classes and interfaces, and establish relationships among the classes and interfaces.

IPO



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

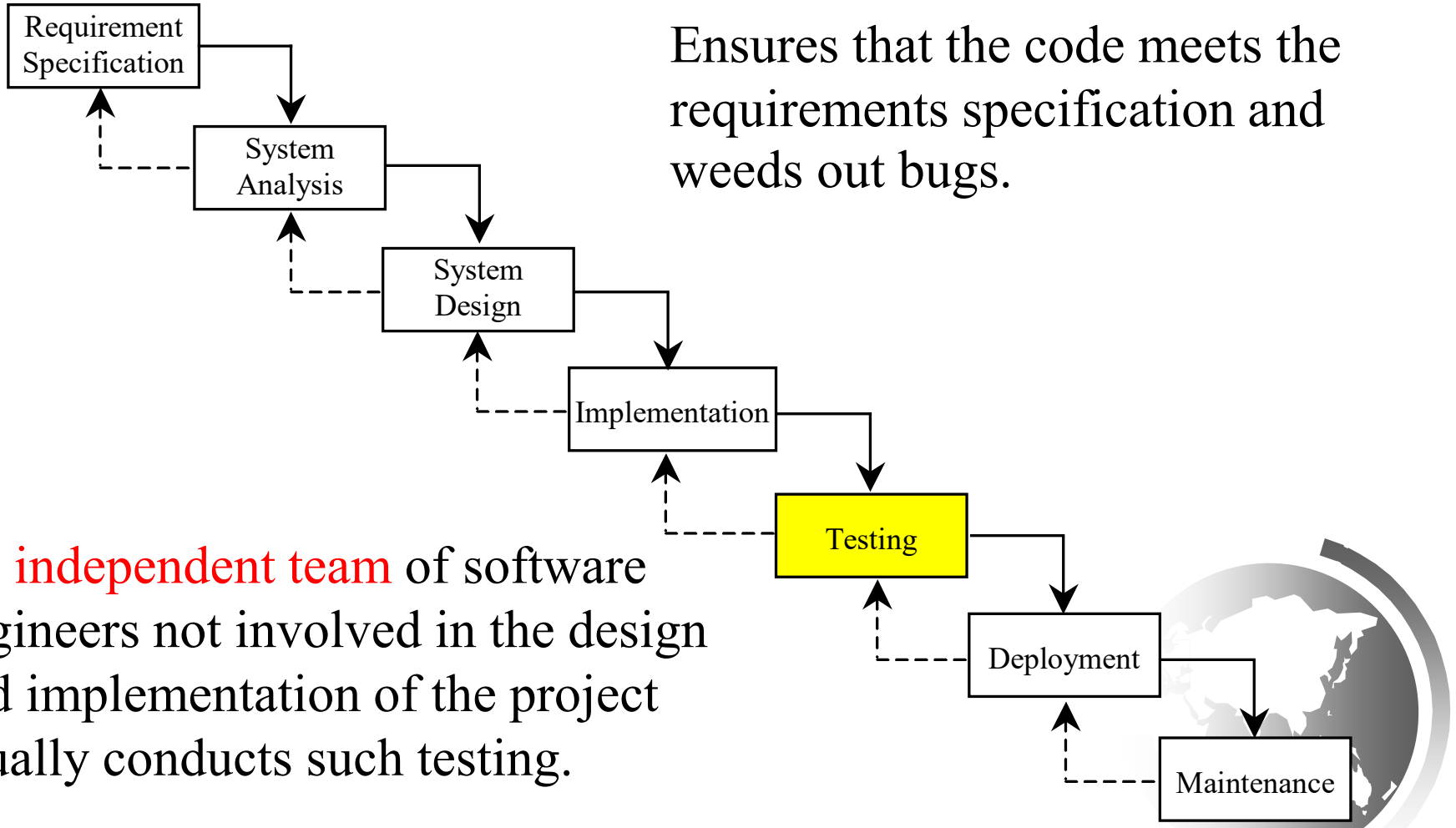
Implementation



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.

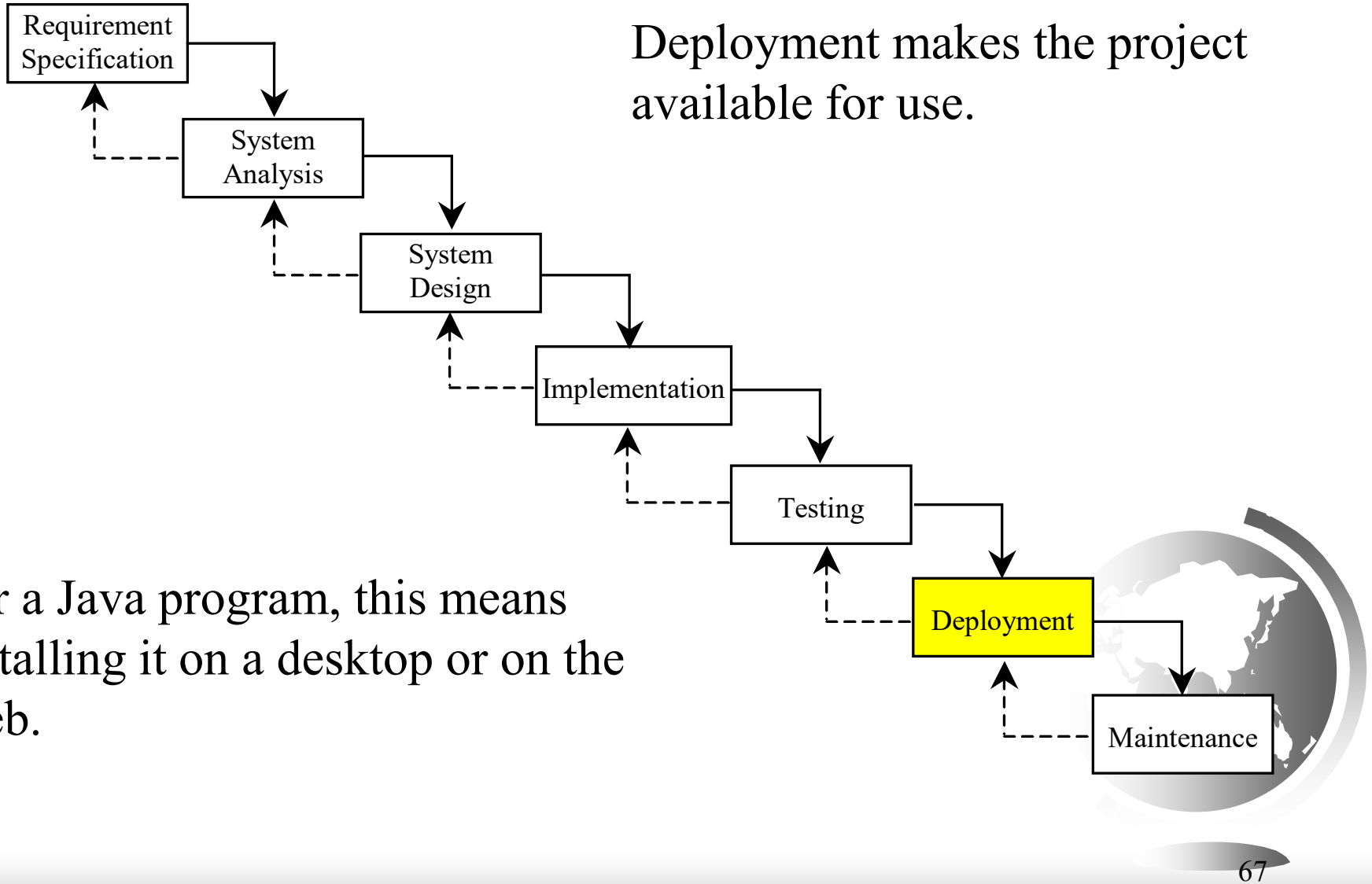
Testing



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.

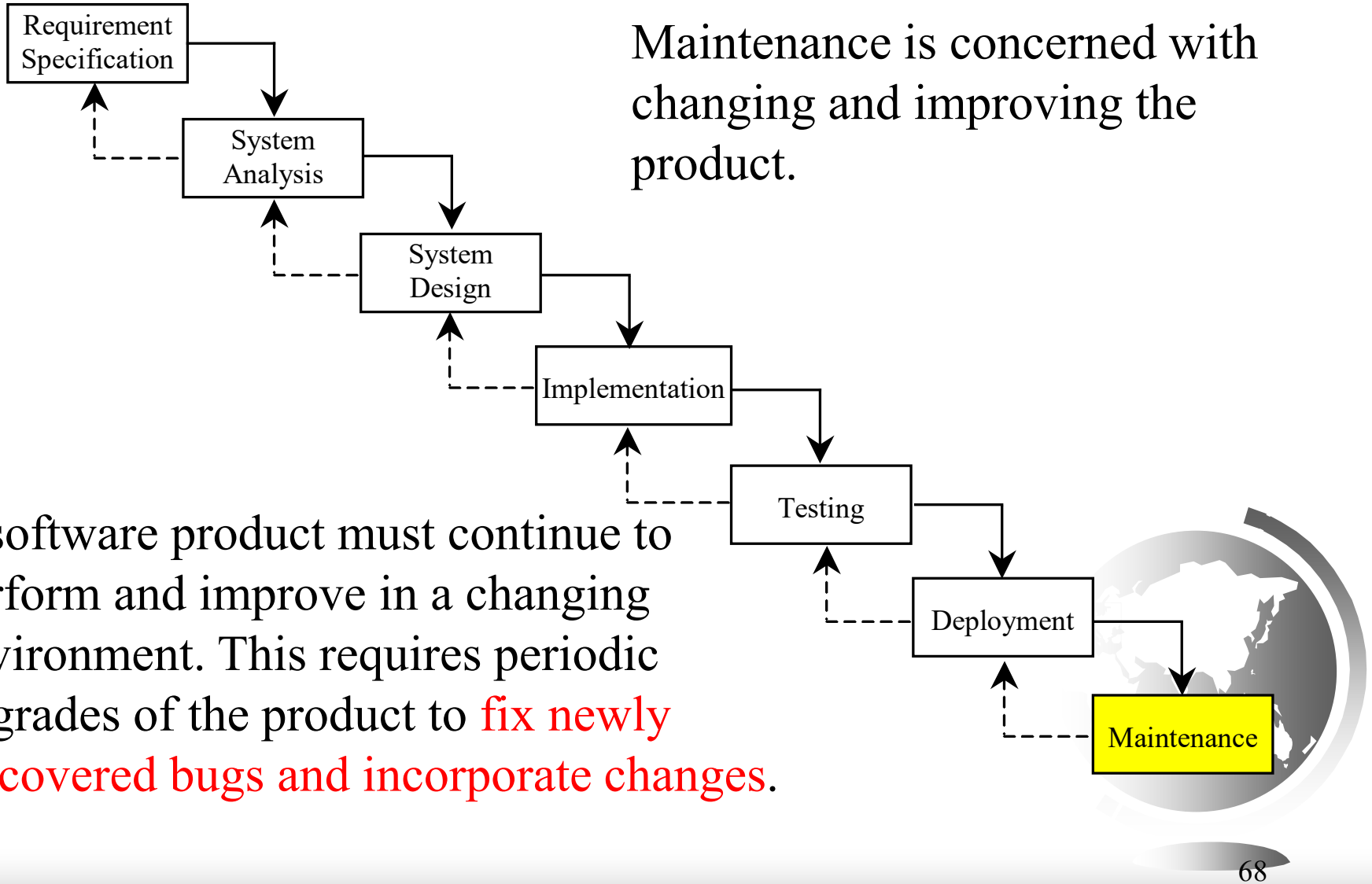
Deployment



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.

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.

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

ComputeLoan

Run

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. Your program should report maximum number of dollars, then the maximum number of quarters, and so on, in this order.

ComputeChange

Run

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;
System.out.println(average);
```

(a)

```
int number1 = 1;
int number2 = 2;
double average = (number1 + number2) / 2.0;
System.out.println(average);
```

(b)

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();
```