CSE 101 and CSE 101T COMPUTER PROGRAMMING I

Joseph LEDET
Department of Computer Engineering
Akdeniz University
josephledet@akdeniz.edu.tr

COURSE SCHEDULE

- Room, Day, & Time:
  - Lecture: Z1 – Tuesday 13:30 – 15:30, Amfi1 – Wednesday 15:30 – 17:30
  - Lab: Yazilim Lab 1 or 3 – Thursday 08:30 or 09:30
- Instructor: Joseph LEDET
  - Office: Computer Engineering Building 2nd Floor
  - E-mail: josephledet@akdeniz.edu.tr
  - Office Hours: Tuesdays 11:30 – 13:30 or by appointment
- Assistants: Berk ERCİN & Barış Doruk BAŞARAN
  - Office: Computer Engineering Building 2nd Floor
  - E-mail: berkercin@akdeniz.edu.tr & dorukbasaran@akdeniz.edu.tr
  - Office Hours: Thursday 13:30 – 15:30 & Friday 12:30-14:30 or by appointment

COURSE OBJECTIVES

- This course is designed for 1st year computer engineering undergraduate students. The students will be introduced to fundamental concepts of computer programming using Java. This course covers basic programming concepts such as variables, data types, iteration, methods, arrays, etc. Upon successful completion of this course, students will be able to:
  - Write, run and debug programs
  - Explain the differences between various data types such as integer, floating point, character
  - Understand conditionals and loops
  - Reduce large problems into smaller sub-problems, implement sub-problems by writing methods
  - Create and manipulate arrays
  - Read from and write to a file

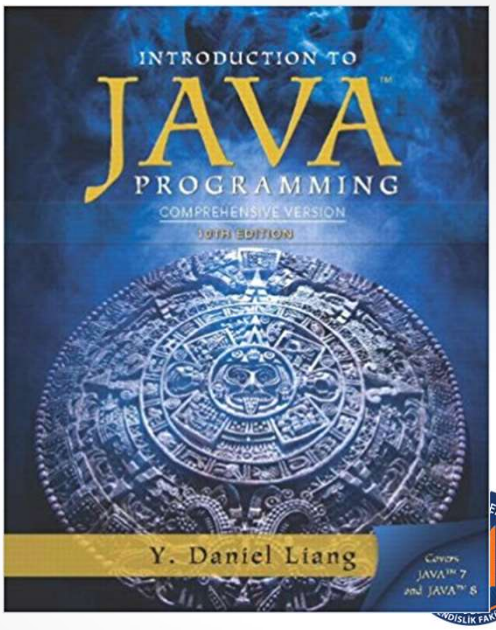
TEXTBOOK

- Y. D. Liang

Introduction to Java Programming, Comprehensive Version (10E)

Pearson, 2014.

ISBN-13: 978-0133761313



COURSE INFORMATION

- The lecture slides, announcements, and homework will be available through Microsoft Teams. Make sure that you have been added to the course **24G_CSE 101T_Computer Programming I_Ö_1**.
  - https://www.microsoft.com/en-us/microsoft-365/microsoft-teams/group-chat-software/
- It is your responsibility to check frequently for updates concerning this course.
- You can install the mobile app (Android or IOS) to your phone so that you can get notifications whenever an announcement is made.

GRADING (Tentative)

CSE 101	CSE 101 T	Weight
Quizzes	Quizzes	10%
Lab Activities	Attendance/Participation	10%
Assignments	Assignments	20%
Midterm	Midterm	20%
Final	Final	40%

ATTENDANCE and PARTICIPATION

- Attendance and class discussions are essential for the learning process
- If you miss a class it is your responsibility to find out what was discussed in the class
  - Ask and answer questions
- A portion of your grade will be based on your contributions to the class discussions
- There will be no make up for quizzes if you miss one

QUIZZES, LAB ASSIGNMENTS, HOMEWORK

- There will be between 1 and 26 quizzes this semester.
- All homework/programming assignments are due by midnight (23:59) on the date set by instructor or TA as the deadline. You will lose 10% for each day they are late and submissions will not be accepted after two days.

EXAMINATIONS

- There will be no make-up for any midterm unless a medical excuse is provided.
- Final exam will be comprehensive and you will be responsible for all the topics covered during the semester.
  - There will be a final re-take exam after the letter grades are posted. This exam will be a replacement exam for your final exam score.

ACADEMIC INTEGRITY

- If you get caught cheating, maximum possible action will be taken, including reporting the matter to the appropriate university authorities.
- Please cooperate by doing your own work and not seeking inappropriate help from your classmates.
- It is encouraged to discuss course topics and assignments amongst yourselves, as long as that discussion does not lead to an exchange of solutions.

COURSE OUTLINE

Week	Wed	Thur	Chapter	Tentative Topics
1	Oct 3	Oct 4	1	Introduction
2	Oct 10	Oct 11	2	Elementary Programming
3	Oct 17	Oct 18	3	Selections
4	Oct 24	Oct 25	4	Math Functions, Character, String
5	Oct 31	Nov 1	5	Loops
6	Nov 7	Nov 8	5	Loops
7	Nov 14	Nov 15	6	Methods
8	Nov 21	Nov 22		Tentative Midterm Week
9	Nov 28	Nov 29	6	Methods
10	Dec 5	Dec 6	7	Arrays
11	Dec 12	Dec 13	8	Arrays
12	Dec 19	Dec 20	9	File I/O
13	Dec 26	Dec 27	10	Recursion
14	Jan 2	Jan 3		Final Review

COMMUNICATION

- If you need to email me, please
  - Use my university email address (josephledet@akdeniz.edu.tr)
  - Begin the subject line with "CSE 101" and include a brief (maximum of a few words) description of your comment or question
    - CSE 101: Midterm Exam
    - CSE 101: Assignment 2
    - CSE 101: Quiz 1
  - I will do everything within my power to respond within 24 hours (business day)
  - Make sure your question has not already been addressed in the course material (slides, announcements, etc.)
  - **DO NOT USE** Google Translate!

IMPORTANT NOTES

- Keep up with the course.
  - Easter Eggs!
  - The first ten students to write me an email or Teams message with the name of the creator of the first compiler, the name of the compiler, and the year it was unveiled will receive a bonus 10 points to their homework total this semester.
- When in doubt…ASK!
- Attempting and failing is better than not attempting at all.
  - Partial credit is your friend!
- Practice!
- If you have a class or lab conflict or if you want to use last year's lab, attendance, or quiz grades, we must be notified by email no later than the end of the third class week (19-Oct-2023).

THIS WEEK

- Install a Java IDE.
- Suggestions
  - JGrasp (http://www.jgrasp.org/)
  - NetBeans (https://netbeans.org/)
  - IntelliJ (https://www.jetbrains.com/idea/)
  - Eclipse (http://www.eclipse.org/)

CSE 101 - COMPUTER PROGRAMMING I INTRODUCTION

Joseph LEDET
Department of Computer Engineering
Akdeniz University
josephledet@akdeniz.edu.tr

WHAT IS ENGINEERING?

- Engineering is…
- Analyzing problems
- Solving problems
- Using tools to create solutions
- If a tool does not exist, an engineer may need to create the tool

- Engineering is not…
- Memorizing
- Answering questions
- Finding the one right answer
- An Engineer does not claim someone else's solution as their own

ALGORITHM

- What is an algorithm?
- Components of an algorithm
- How to express algorithms

SOLVING PROBLEMS

- How to solve a problem:
  - Brute Force
  - Lucky Guess
  - Trial-and-Error
  - Experience (your own or someone else's)
  - Scientifically/Algorithmically

ALGORITHM

- Sequence of instructions describing how to accomplish a task
  - Note: Not actually performing the instructions
- Examples
  - Cooking recipe
  - Assembly instructions
  - Directions for driving from Antalya to Ankara
  - Instructions for installing software (such as Java)

INSTALL SOFTWARE ALGORITHM

- Open web browser
- Go to
  - https://www.oracle.com/java/technologies/javase-jdk15-downloads.html
- Click "jdk-15_windows-x64_bin.exe"
- Follow instructions

EXAMPLE – MAKING AYRAN

- Put 500 grams of yogurt into a mixer
- Add ½ liter water
- Add some salt
- Turn on mixer until contents are smooth
- Enjoy!

COMPONENTS OF AN ALGORITHM

- Values and variables (500 grams of yogurt)
- Instructions (Add ½ liter water)
- Sequences (Step 1 then 2)
- Procedures (Turn on mixer)
- Selections (if some condition is true, do something)
- Repetitions (… until smooth)
- Documentation

VALUES AND VARIABLES

- Values
  - Represent quantities, amounts, or measurements
  - May be one of many different types (number, text, other)
- Variables
  - Containers for values
  - Example
    - This jar (Variable) can contain 10 cookies (Value)

INSTRUCTIONS AND SEQUENCE

- Instructions
  - Directions to perform an action on values and variables
  - Examples
    - Put (instruction) 500 grams of yogurt into the mixer
    - Add (instruction) ½ liter of water
- Sequence
  - The order of a program is important
  - What if we performed the corrected ayran recipe in reverse order?

PROCEDURE

- A defined and named sequence of instructions
- Can refer to it by name rather than repeating the same instructions many times
- Examples
  - Mix – can include instructions to close the lid, make sure it is plugged in, press the button, etc.
  - Walk

SELECTION

- Instruction to decide which of two possible sequences is performed
- Based on a true/false condition
- Example
  - If I like ayran (condition), then drink (one sequence), otherwise, say "No thank you" (other sequence)
  - If class is over, then leave the meeting and enjoy the weekend, otherwise, remain online
- Can have multiple conditions
  - If class is over AND I don't have questions…

REPETITION

- Similar to selection but as long as condition is true, repeat sequence
- Test the condition, if true perform instructions, repeat…
- Also known as iteration or loop
- Example
  - Mix (instruction) until (repetition keyword) smooth (condition)
  - While (repetition keyword) teacher is speaking (condition) pay attention and take notes (instruction)

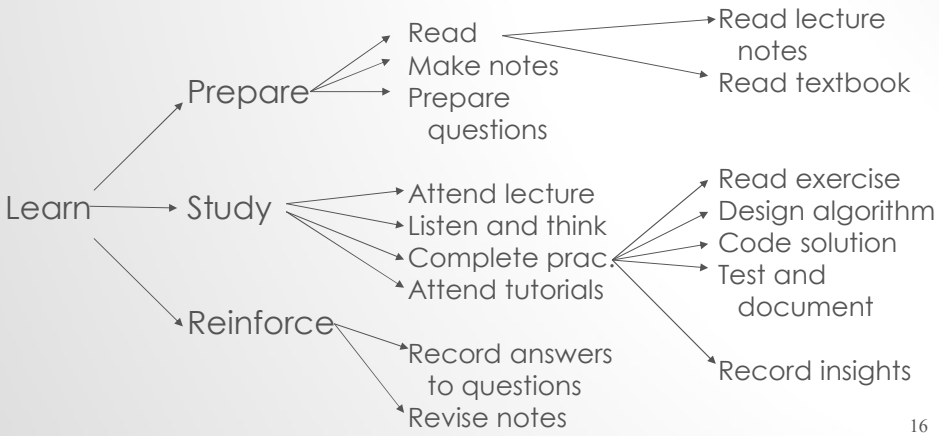
DOCUMENTATION

- Records what the algorithm does
- Describes how it is accomplished
- Explains the purpose of components
- Notes restrictions or expectations
- Example
  - "A recipe for making authentic, tasty ayran"
  - "Taking notes is a terrific way to learn the information presented"

ALGORITHM DESIGN

- Start with the desired goal of the algorithm
- Divide into smaller activities
- Divide these into even smaller activities until each resulting activity is easily performed (possibly in a single instruction)
- Example
  - Learning

LEARNING



EXPRESSING ALGORITHMS

- Natural Language
- Pseudocode
- Flowcharts
- Programming Languages

PSEUDOCODE

- If student's grade is greater than or equal to 90
  - Print "A"
- Else If student's grade is greater than or equal to 80
  - Print "B"
- Else If student's grade is greater than or equal to 70
  - Print "C"
- Else If student's grade is greater than or equal to 60
  - Print "D"
- Else
  - Print "F"

PSEUDOCODE

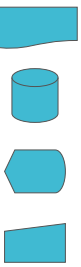
- Set total to 0
- Set classCount to 0
- While more grades exist to input
  - Input the next grade
  - Add grade to total (total = total + grade)
  - Increment classCount (classCount = classCount + 1)
- Set classAverage to total divided by classCount
  - (classAverage = total / classCount)

FLOWCHARTS

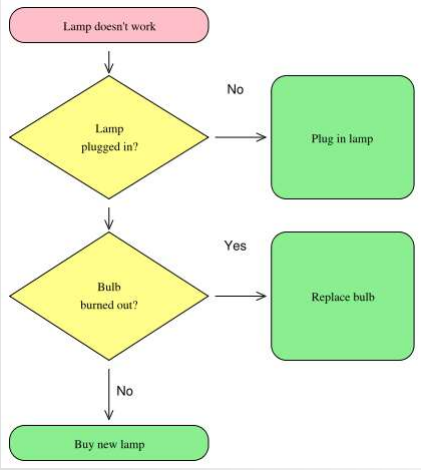
- Visual representation of an algorithm
- Symbols mostly used
  - Start/End
  - Arrows
  - Instructions
  - Input/Output
  - Conditional (Decision)

Less used

- Document
- Magnetic Tape
- Display
- Manual Input

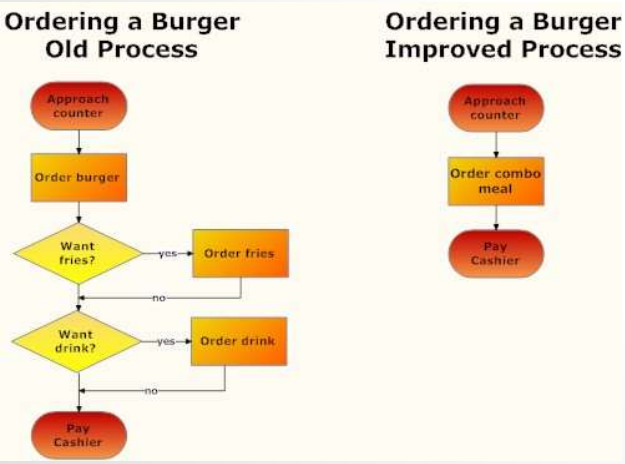


FLOWCHART EXAMPLE



From https://en.wikipedia.org/wiki/Flowchart

FLOWCHART EXAMPLE



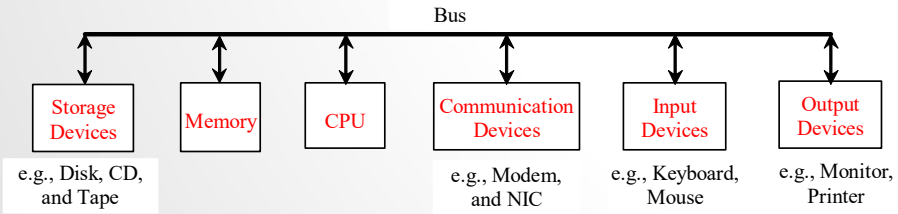
From http://www.smartdraw.com/

Objectives

- To understand computer basics, programs, and operating systems (§§1.2–1.4).
- To describe the relationship between Java and the World Wide Web (§1.5).
- To understand the meaning of Java language specification, API, JDK, and IDE (§1.6).
- To write a simple Java program (§1.7).
- To display output on the console (§1.7).
- To explain the basic syntax of a Java program (§1.7).
- To create, compile, and run Java programs (§1.8).
- To use sound Java programming style and document programs properly (§1.9).
- To explain the differences between syntax errors, runtime errors, and logic errors (§1.10).
- To develop Java programs using NetBeans (§1.11).
- To develop Java programs using Eclipse (§1.12).

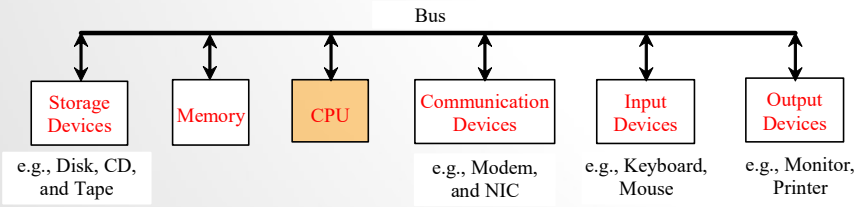
What is a Computer?

A computer consists of a CPU, memory, hard disk, floppy disk, monitor, printer, and communication devices.



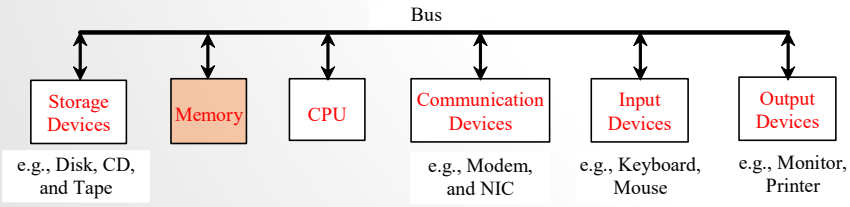
CPU

The central processing unit (CPU) is the brain of a computer. It retrieves instructions from memory and executes them. The CPU speed is measured in megahertz (MHz), with 1 megahertz equaling 1 million pulses per second. The speed of the CPU has been improved continuously. If you buy a PC now, you can get an Intel Pentium 4 Processor at 3 gigahertz (1 gigahertz is 1000 megahertz).



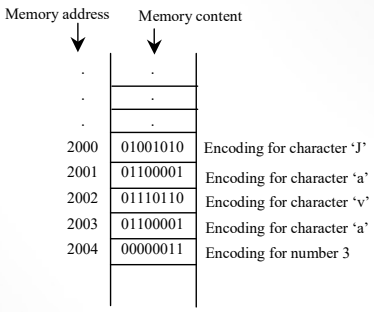
Memory

Memory is to store data and program instructions for CPU to execute. A memory unit is an ordered sequence of bytes, each holds eight bits. A program and its data must be brought to memory before they can be executed. A memory byte is never empty, but its initial content may be meaningless to your program. The current content of a memory byte is lost whenever new information is placed in it.



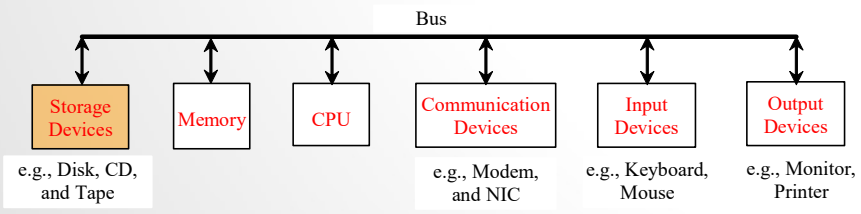
How Data is Stored?

Data of various kinds, such as numbers, characters, and strings, are encoded as a series of bits (zeros and ones). Computers use zeros and ones because digital devices have two stable states, which are referred to as *zero* and *one* by convention. The programmers need not to be concerned about the encoding and decoding of data, which is performed automatically by the system based on the encoding scheme. The encoding scheme varies. For example, character 'J' is represented by 01001010 in one byte. A small number such as three can be stored in a single byte. If computer needs to store a large number that cannot fit into a single byte, it uses a number of adjacent bytes. No two data can share or split a same byte. A byte is the minimum storage unit.



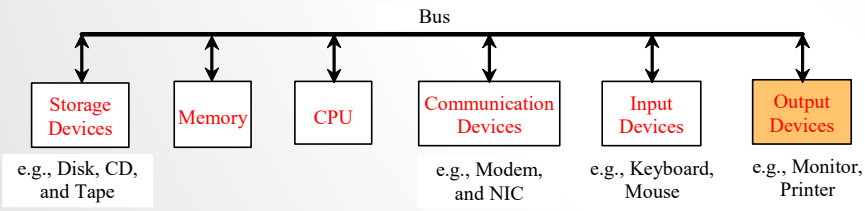
Storage Devices

Memory is volatile, because information is lost when the power is off. Programs and data are permanently stored on storage devices and are moved to memory when the computer actually uses them. There are three main types of storage devices:Disk drives (hard disks and floppy disks), CD drives (CD-R and CD-RW), and Tape drives.



Output Devices: Monitor

The monitor displays information (text and graphics). The resolution and dot pitch determine the quality of the display.



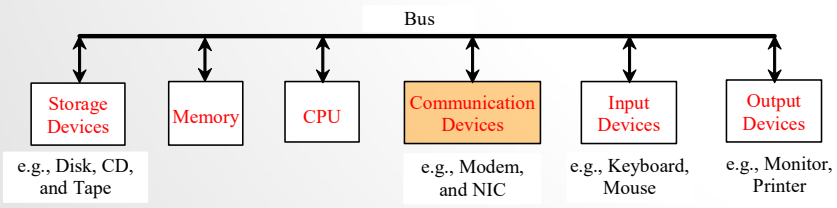
Monitor Resolution and Dot Pitch

resolution The *screen resolution* specifies the number of pixels in horizontal and vertical dimensions of the display device. *Pixels* (short for "picture elements") are tiny dots that form an image on the screen. A common resolution for a 17-inch screen, for example, is 1,024 pixels wide and 768 pixels high. The resolution can be set manually. The higher the resolution, the sharper and clearer the image is.

dot pitch The *dot pitch* is the amount of space between pixels, measured in millimeters. The smaller the dot pitch, the sharper the display.

Communication Devices

A *regular modem* uses a phone line and can transfer data in a speed up to 56,000 bps (bits per second). A *DSL* (digital subscriber line) also uses a phone line and can transfer data in a speed 20 times faster than a regular modem. A *cable modem* uses the TV cable line maintained by the cable company. A cable modem is as fast as a DSL. Network interface card (*NIC*) is a device to connect a computer to a local area network (LAN). The LAN is commonly used in business, universities, and government organizations. A typical type of NIC, called *10BaseT*, can transfer data at 10 mbps (million bits per second).



Programs

Computer *programs*, known as *software*, are instructions to the computer.

You tell a computer what to do through programs. Without programs, a computer is an empty machine. Computers do not understand human languages, so you need to use computer languages to communicate with them.

Programs are written using programming languages.

Programming Languages

Machine Language Assembly Language High-Level Language

Machine language is a set of primitive instructions built into every computer. The instructions are in the form of binary code, so you have to enter binary codes for various instructions. Program with native machine language is a tedious process. Moreover the programs are highly difficult to read and modify. For example, to add two numbers, you might write an instruction in binary like this:

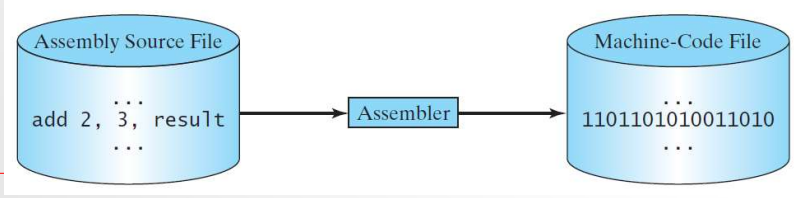
```
1101101010011010
```


Programming Languages

Machine Language Assembly Language High-Level Language

Assembly languages were developed to make programming easy. Since the computer cannot understand assembly language, however, a program called assembler is used to convert assembly language programs into machine code. For example, to add two numbers, you might write an instruction in assembly code like this:

```
ADDF3 R1, R2, R3
```



Programming Languages

Machine Language Assembly Language High-Level Language

The high-level languages are English-like and easy to learn and program. For example, the following is a high-level language statement that computes the area of a circle with radius 5:

```
area = 5 * 5 * 3.1415;
```


Popular High-Level Languages

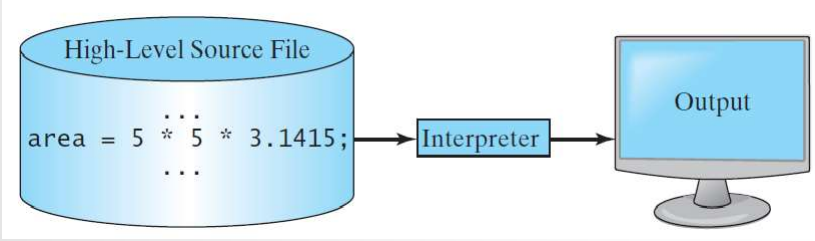
Language	Description
Ada	Named for Ada Lovelace, who worked on mechanical general-purpose computers. The Ada language was developed for the Department of Defense and is used mainly in defense projects.
BASIC	Beginner's All-purpose Symbolic Instruction Code. It was designed to be learned and used easily by beginners.
C	Developed at Bell Laboratories. C combines the power of an assembly language with the ease of use and portability of a high-level language.
C++	C++ is an object-oriented language, based on C.
C#	Pronounced "C Sharp." It is a hybrid of Java and C++ and was developed by Microsoft.
COBOL	COmmon Business Oriented Language. Used for business applications.
FORTRAN	FORmula TRANslation. Popular for scientific and mathematical applications.
Java	Developed by Sun Microsystems, now part of Oracle. It is widely used for developing platform-independent Internet applications.
Pascal	Named for Blaise Pascal, who pioneered calculating machines in the seventeenth century. It is a simple, structured, general-purpose language primarily for teaching programming.
Python	A simple general-purpose scripting language good for writing short programs.
Visual Basic	Visual Basic was developed by Microsoft and it enables the programmers to rapidly develop graphical user interfaces.

Interpreting/Compiling Source Code

A program written in a high-level language is called a *source program* or *source code*. Because a computer cannot understand a source program, a source program must be translated into machine code for execution. The translation can be done using another programming tool called an *interpreter* or a *compiler*.

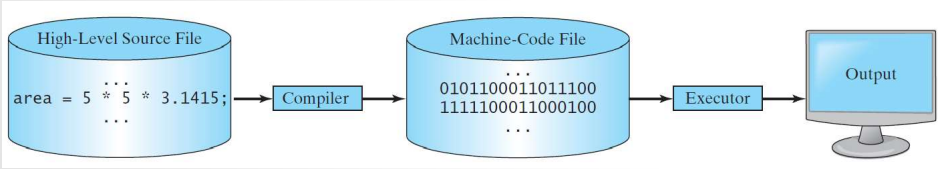
Interpreting Source Code

An interpreter reads one statement from the source code, translates it to the machine code or virtual machine code, and then executes it right away, as shown in the following figure. Note that a statement from the source code may be translated into several machine instructions.



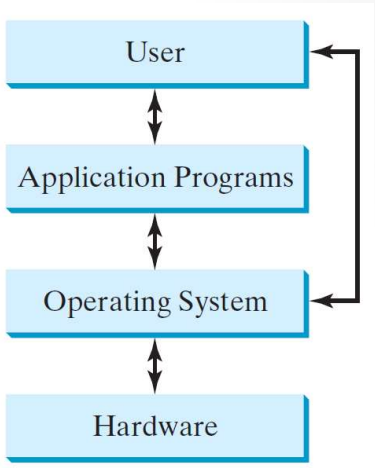
Compiling Source Code

A compiler translates the entire source code into a machine-code file, and the machine-code file is then executed, as shown in the following figure.



Operating Systems

The *operating system* (OS) is a program that manages and controls a computer's activities. The popular operating systems for general-purpose computers are Microsoft Windows, Mac OS, and Linux. Application programs, such as a Web browser or a word processor, cannot run unless an operating system is installed and running on the computer.



Why Java?

The answer is that Java enables users to develop and deploy applications on the Internet for servers, desktop computers, and small hand-held devices. The future of computing is being profoundly influenced by the Internet, and Java promises to remain a big part of that future. Java is the Internet programming language.

- Java is a general purpose programming language.
- Java is the Internet programming language.

Java, Web, and Beyond

- Java can be used to develop standalone applications.
- Java can be used to develop applications running from a browser.
- Java can also be used to develop applications for hand-held devices.
- Java can be used to develop applications for Web servers.

Java's History

- James Gosling and Sun Microsystems
- Oak
- Java, May 20, 1995, Sun World
- HotJava
  - The first Java-enabled Web browser
- Early History Website:

http://www.java.com/en/javahistory/index.jsp

Companion Website

Characteristics of Java

- Java Is Simple
- Java Is Object-Oriented
- Java Is Distributed
- Java Is Interpreted
- Java Is Robust
- Java Is Secure
- Java Is Architecture-Neutral
- Java Is Portable
- Java's Performance
- Java Is Multithreaded
- Java Is Dynamic

www.cs.armstrong.edu/liang/JavaCharacteristics.pdf

Companion Website

Characteristics of Java

- Java Is Simple
- Java Is Object-Oriented
- Java Is Distributed
- Java Is Interpreted
- Java Is Robust
- Java Is Secure
- Java Is Architecture-Neutral
- Java Is Portable
- Java's Performance
- Java Is Multithreaded
- Java Is Dynamic

Java is partially modeled on C++, but greatly simplified and improved. Some people refer to Java as "C++--" because it is like C++ but with more functionality and fewer negative aspects.

Companion Website

Characteristics of Java

- Java Is Simple
- Java Is Object-Oriented
- Java Is Distributed
- Java Is Interpreted
- Java Is Robust
- Java Is Secure
- Java Is Architecture-Neutral
- Java Is Portable
- Java's Performance
- Java Is Multithreaded
- Java Is Dynamic

Java is inherently object-oriented. Although many object-oriented languages began strictly as procedural languages, Java was designed from the start to be object-oriented. Object-oriented programming (OOP) is a popular programming approach that is replacing traditional procedural programming techniques.

One of the central issues in software development is how to reuse code. Object-oriented programming provides great flexibility, modularity, clarity, and reusability through encapsulation, inheritance, and polymorphism.

Companion Website

Characteristics of Java

- Java Is Simple
- Java Is Object-Oriented
- Java Is Distributed
- Java Is Interpreted
- Java Is Robust
- Java Is Secure
- Java Is Architecture-Neutral
- Java Is Portable
- Java's Performance
- Java Is Multithreaded
- Java Is Dynamic

Distributed computing involves several computers working together on a network. Java is designed to make distributed computing easy. Since networking capability is inherently integrated into Java, writing network programs is like sending and receiving data to and from a file.

47

Companion Website

Characteristics of Java

- Java Is Simple
- Java Is Object-Oriented
- Java Is Distributed
- Java Is Interpreted
- Java Is Robust
- Java Is Secure
- Java Is Architecture-Neutral
- Java Is Portable
- Java's Performance
- Java Is Multithreaded
- Java Is Dynamic

You need an interpreter to run Java programs. The programs are compiled into the Java Virtual Machine code called bytecode. The bytecode is machine-independent and can run on any machine that has a Java interpreter, which is part of the Java Virtual Machine (JVM).

48

Companion Website

Characteristics of Java

- Java Is Simple
- Java Is Object-Oriented
- Java Is Distributed
- Java Is Interpreted
- Java Is Robust
- Java Is Secure
- Java Is Architecture-Neutral
- Java Is Portable
- Java's Performance
- Java Is Multithreaded
- Java Is Dynamic

Java compilers can detect many problems that would first show up at execution time in other languages.

Java has eliminated certain types of error-prone programming constructs found in other languages.

Java has a runtime exception-handling feature to provide programming support for robustness.

Companion Website

Characteristics of Java

- Java Is Simple
- Java Is Object-Oriented
- Java Is Distributed
- Java Is Interpreted
- Java Is Robust
- Java Is Secure
- Java Is Architecture-Neutral
- Java Is Portable
- Java's Performance
- Java Is Multithreaded
- Java Is Dynamic

Java implements several security mechanisms to protect your system against harm caused by stray programs.

Companion Website

Characteristics of Java

- Java Is Simple
- Java Is Object-Oriented
- Java Is Distributed
- Java Is Interpreted
- Java Is Robust
- Java Is Secure
- Java Is Architecture-Neutral
- Java Is Portable
- Java's Performance
- Java Is Multithreaded
- Java Is Dynamic

Write once, run anywhere

With a Java Virtual Machine (JVM), you can write one program that will run on any platform.

Companion Website

Characteristics of Java

- Java Is Simple
- Java Is Object-Oriented
- Java Is Distributed
- Java Is Interpreted
- Java Is Robust
- Java Is Secure
- Java Is Architecture-Neutral
- Java Is Portable
- Java's Performance
- Java Is Multithreaded
- Java Is Dynamic

Because Java is architecture neutral, Java programs are portable. They can be run on any platform without being recompiled.

Companion Website

Characteristics of Java

- Java Is Simple
- Java Is Object-Oriented
- Java Is Distributed
- Java Is Interpreted
- Java Is Robust
- Java Is Secure
- Java Is Architecture-Neutral
- Java Is Portable
- Java's Performance
- Java Is Multithreaded
- Java Is Dynamic

Java's performance Because Java is architecture neutral, Java programs are portable. They can be run on any platform without being recompiled.

Companion Website

Characteristics of Java

- Java Is Simple
- Java Is Object-Oriented
- Java Is Distributed
- Java Is Interpreted
- Java Is Robust
- Java Is Secure
- Java Is Architecture-Neutral
- Java Is Portable
- Java's Performance
- Java Is Multithreaded
- Java Is Dynamic

Multithread programming is smoothly integrated in Java, whereas in other languages you have to call procedures specific to the operating system to enable multithreading.

Companion Website

Characteristics of Java

- Java Is Simple
- Java Is Object-Oriented
- Java Is Distributed
- Java Is Interpreted
- Java Is Robust
- Java Is Secure
- Java Is Architecture-Neutral
- Java Is Portable
- Java's Performance
- Java Is Multithreaded
- Java Is Dynamic

Java was designed to adapt to an evolving environment. New code can be loaded on the fly without recompilation. There is no need for developers to create, and for users to install, major new software versions. New features can be incorporated transparently as needed.

JDK Editions

- Java Standard Edition (J2SE)
  - J2SE can be used to develop client-side standalone applications or applets.
- Java Enterprise Edition (J2EE)
  - J2EE can be used to develop server-side applications such as Java servlets, Java ServerPages, and Java ServerFaces.
- Java Micro Edition (J2ME).
  - J2ME can be used to develop applications for mobile devices such as cell phones.

This book uses J2SE to introduce Java programming.

Popular Java IDEs

- NetBeans
- Eclipse
- IntelliJ
- jGrasp

A Simple Java Program

Listing 1.1

```
// This program prints Welcome to Java!
public class Welcome {
  public static void main(String[] args) {
    System.out.println("Welcome to Java!");
  }
}
```

Animation

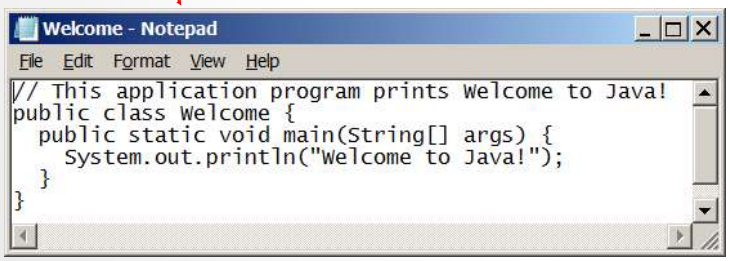
Welcome

Run

IMPORTANT NOTE: If you cannot run the buttons, see www.cs.armstrong.edu/liang/javaslidenote.doc.

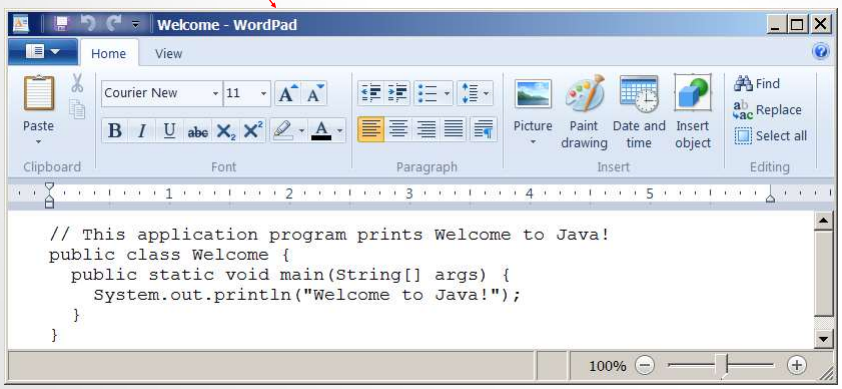
Creating and Editing Using NotePad

To use NotePad, type
notepad Welcome.java
from the DOS prompt.

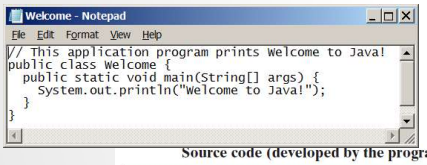


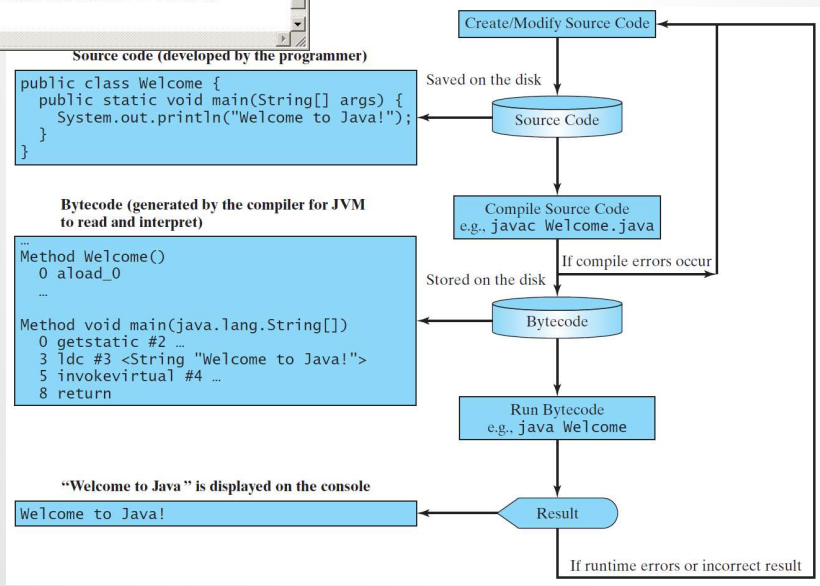
Creating and Editing Using WordPad

To use WordPad, type
write Welcome.java
from the DOS prompt.



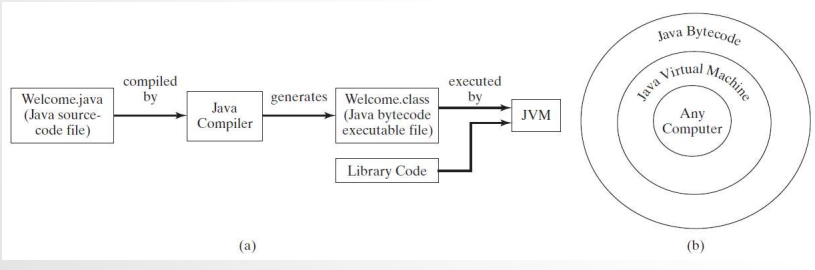
Creating, Compiling, and Running Programs





Compiling Java Source Code

You can port a source program to any machine with appropriate compilers. The source program must be recompiled, however, because the object program can only run on a specific machine. Nowadays computers are networked to work together. Java was designed to run object programs on any platform. With Java, you write the program once, and compile the source program into a special type of object code, known as *bytecode*. The bytecode can then run on any computer with a Java Virtual Machine, as shown below. Java Virtual Machine is a software that interprets Java bytecode.



animation

Trace a Program Execution

Enter main method

```
// This program prints Welcome to Java!
public class Welcome {
  public static void main(String[] args) {
    System.out.println("Welcome to Java!");
  }
}
```

animation

Trace a Program Execution

Execute statement

```
// This program prints Welcome to Java!
public class Welcome {
  public static void main(String[] args) {
    System.out.println("Welcome to Java!");
  }
}
```

animation

Trace a Program Execution

```
// This program prints Welcome to Java!
public class Welcome {
  public static void main(String[] args) {
    System.out.println("Welcome to Java!");
  }
}
```

print a message to the console

Two More Simple Examples

Animation

WelcomeWithThreeMessages Run

Animation

ComputeExpression Run

Companion Website

Supplements on the Companion Website

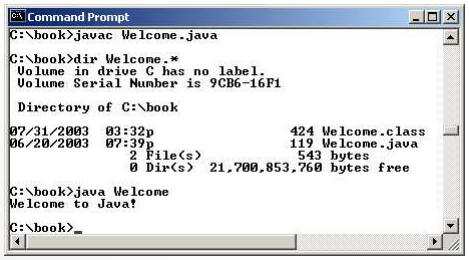
- See Supplement I.B for installing and configuring JDK
- See Supplement I.C for compiling and running Java from the command window for details

www.cs.armstrong.edu/liang/intro10e

Companion Website

Compiling and Running Java from the Command Window

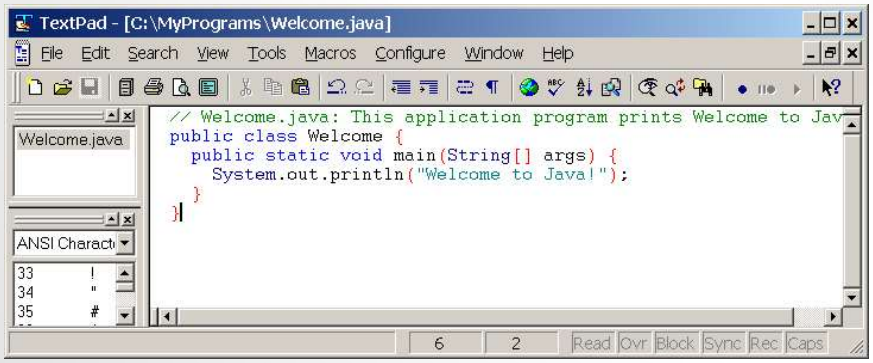
- Set path to JDK bin directory
  - set path=c:\Program Files\java\jdk-15\bin
- Set classpath to include the current directory
  - set classpath=.
- Compile
  - javac Welcome.java
- Run
  - java Welcome



Compiling and Running Java from TextPad

Companion Website

- See Supplement II.A on the Website for details



Anatomy of a Java Program

- Class name
- Main method
- Statements
- Statement terminator
- Reserved words
- Comments
- Blocks

Class Name

Every Java program must have at least one class. Each class has a name. By convention, class names start with an uppercase letter. In this example, the class name is Welcome.

```
// This program prints Welcome to Java!
public class Welcome {
  public static void main(String[] args) {
    System.out.println("Welcome to Java!");
  }
}
```


Main Method

Line 2 defines the main method. In order to run a class, the class must contain a method named main. The program is executed from the main method.

```
// This program prints Welcome to Java!
public class Welcome {
  public static void main(String[] args) {
    System.out.println("Welcome to Java!");
  }
}
```


Statement

A statement represents an action or a sequence of actions. The statement System.out.println("Welcome to Java!") in the program in Listing 1.1 is a statement to display the greeting "Welcome to Java!".

```
// This program prints Welcome to Java!
public class Welcome {
  public static void main(String[] args) {
    System.out.println("Welcome to Java!");
  }
}
```


Statement Terminator

Every statement in Java ends with a semicolon (;).

```
// This program prints Welcome to Java!
public class Welcome {
  public static void main(String[] args) {
    System.out.println("Welcome to Java!");
  }
}
```

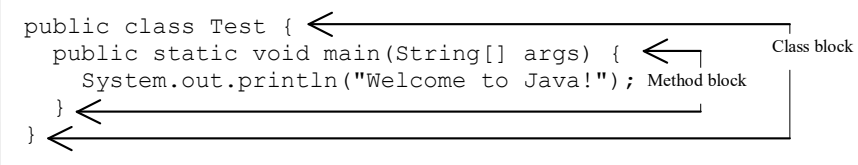

Reserved words

Reserved words or keywords are words that have a specific meaning to the compiler and cannot be used for other purposes in the program. For example, when the compiler sees the word class, it understands that the word after class is the name for the class.

```
// This program prints Welcome to Java!
public class Welcome {
  public static void main(String[] args) {
    System.out.println("Welcome to Java!");
  }
}
```


Blocks

A pair of braces in a program forms a block that groups components of a program.



Special Symbols

Character	Name	Description
{}	Opening and closing braces	Denotes a block to enclose statements.
()	Opening and closing parentheses	Used with methods.
[]	Opening and closing brackets	Denotes an array.
//	Double slashes	Precedes a comment line.
" "	Opening and closing quotation marks	Enclosing a string (i.e., sequence of characters).
;	Semicolon	Marks the end of a statement.

{ ... }

```
// This program prints Welcome to Java!
public class Welcome {
  public static void main(String[] args) {
    System.out.println("Welcome to Java!");
  }
}
```

(...)

```
// This program prints Welcome to Java!
public class Welcome {
  public static void main(String[] args) {
    System.out.println("Welcome to Java!");
  }
}
```


;

```
// This program prints Welcome to Java!
public class Welcome {
  public static void main(String[] args) {
    System.out.println("Welcome to Java!");
  }
}
```


// ...

```
// This program prints Welcome to Java!
public class Welcome {
  public static void main(String[] args) {
    System.out.println("Welcome to Java!");
  }
}
```


" ... "

```
// This program prints Welcome to Java!
public class Welcome {
  public static void main(String[] args) {
    System.out.println(""Welcome to Java!"");
  }
}
```


Programming Style and Documentation

- Appropriate Comments
- Naming Conventions
- Proper Indentation and Spacing Lines
- Block Styles

Appropriate Comments

Include a summary at the beginning of the program to explain what the program does, its key features, its supporting data structures, and any unique techniques it uses.

Include your name, class section, instructor, date, and a brief description at the beginning of the program.

Naming Conventions

- Choose meaningful and descriptive names.
- Class names:
  - Capitalize the first letter of each word in the name. For example, the class name `ComputeExpression`.

Proper Indentation and Spacing

- Indentation
  - Indent two spaces.
- Spacing
  - Use blank line to separate segments of the code.

Block Styles

Use end-of-line style for braces.

Next-line style

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

End-of-line style

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


Programming Errors

- Syntax Errors
  - Detected by the compiler
- Runtime Errors
  - Causes the program to abort
- Logic Errors
  - Produces incorrect result

CSE 101 - COMPUTER PROGRAMMING I ELEMENTARY PROGRAMMING

Joseph LEDET
Department of Computer Engineering
Akdeniz University
josephledet@akdeniz.edu.tr

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.

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

IMPORTANT NOTE: If you cannot run the buttons, see www.cs.armstrong.edu/liang/javaslidenote.doc

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);
  }
}
```

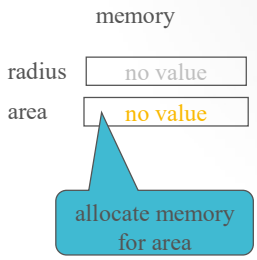
allocate memory for radius

radius | no value

animation

TRACE A PROGRAM EXECUTION

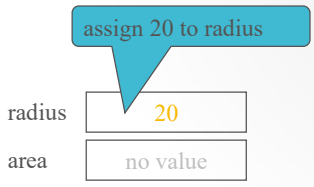
```
public class ComputeArea {
  /** Main method */
  public static void main(String[] args) {
    double radius;
    double area;

    // 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);
  }
}
```

memory

radius	20
area	1256.636

compute area and assign it to variable area

animation

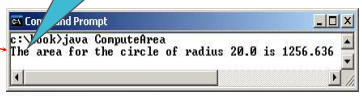
TRACE A PROGRAM EXECUTION

```
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

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.

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;`

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

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

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 %

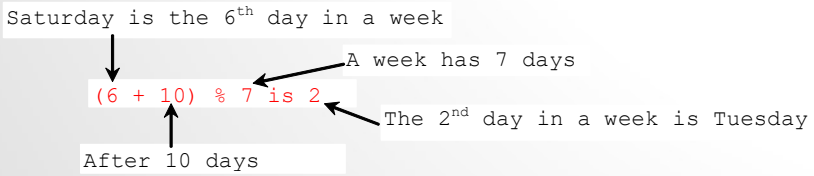
5 / 2 yields an integer 2.

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

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.

```
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:

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

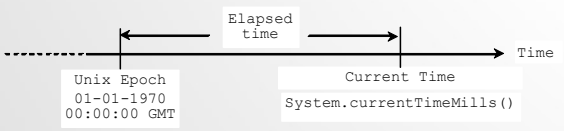
Note: you have to write
celsius = (5.0 / 9) * (fahrenheit – 32)

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.



ShowCurrentTime Run

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

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;

range increases

byte, short, int, long, float, double

PROBLEM: KEEPING TWO DIGITS AFTER DECIMAL POINTS

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

SalesTax

Run

CASTING IN AN AUGMENTED EXPRESSION

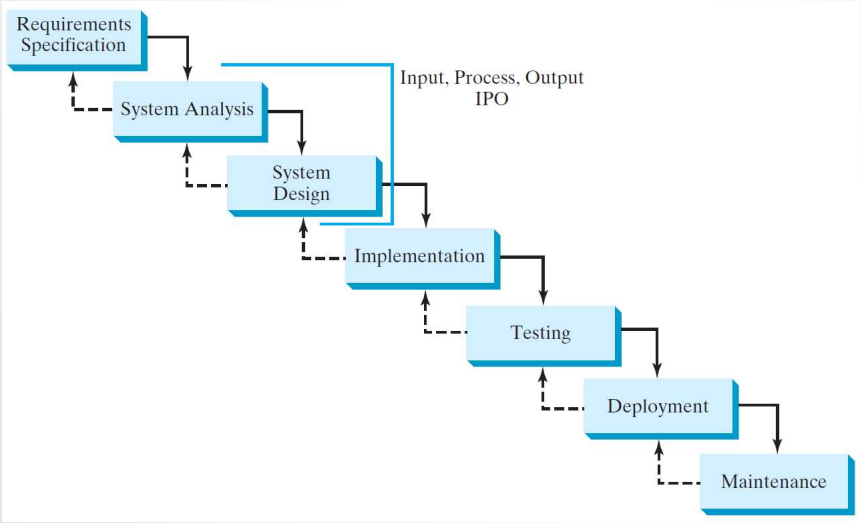
In Java, an augmented expression of the form **x1 op= x2** is implemented as **x1 = (T)(x1 op x2)**, where **T** is the type for **x1**. Therefore, the following code is correct.

int sum = **0**;

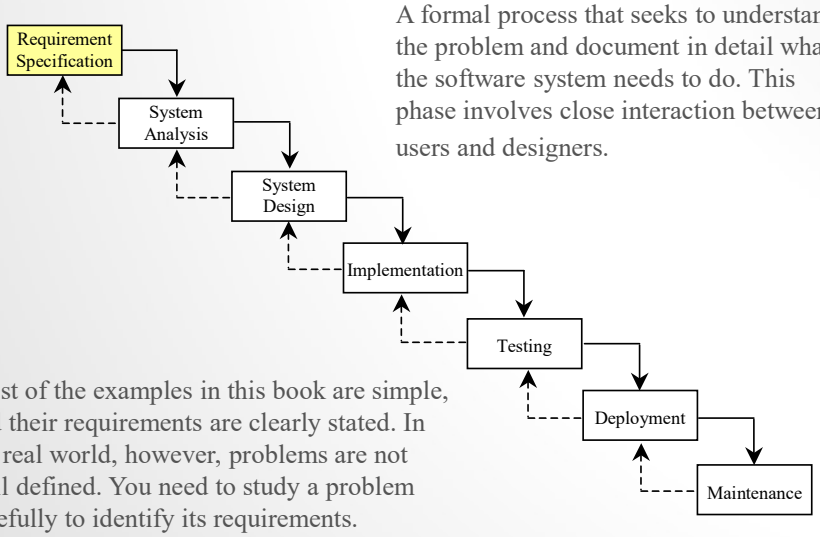
sum += **4.5**; // sum becomes 4 after this statement

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

SOFTWARE DEVELOPMENT PROCESS



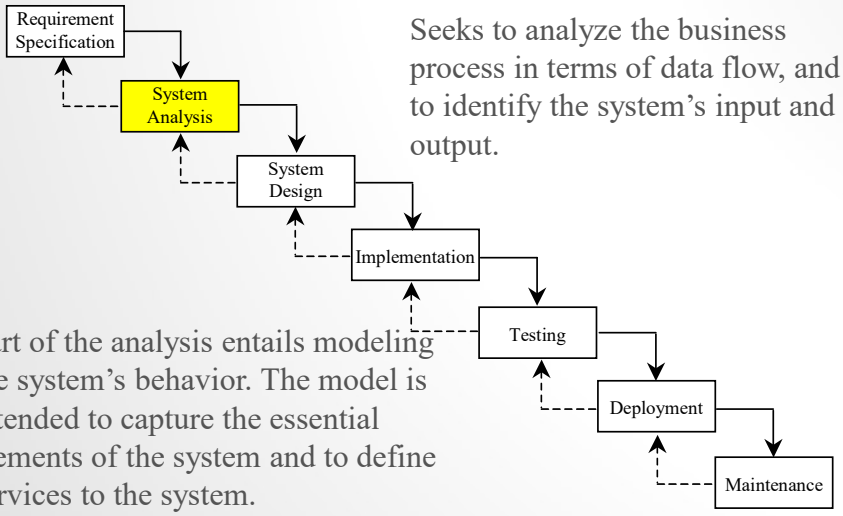
REQUIREMENT SPECIFICATION



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.

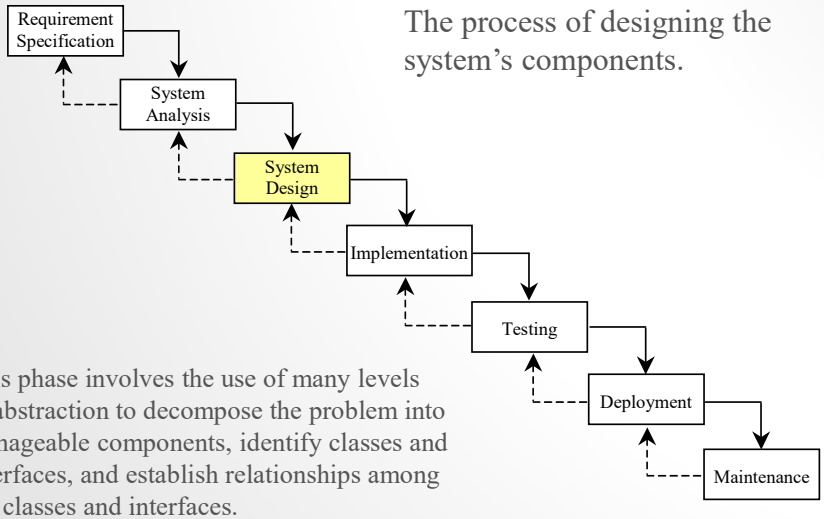
SYSTEM ANALYSIS



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.

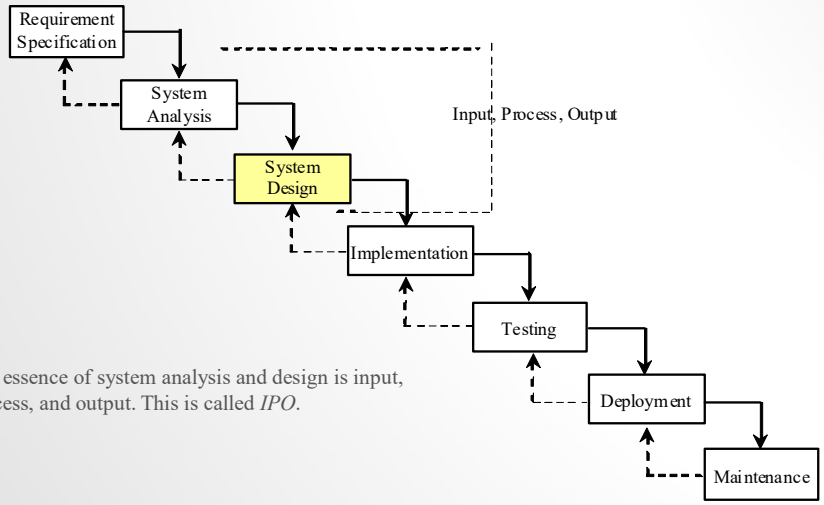
SYSTEM DESIGN



The process of designing the system's components.

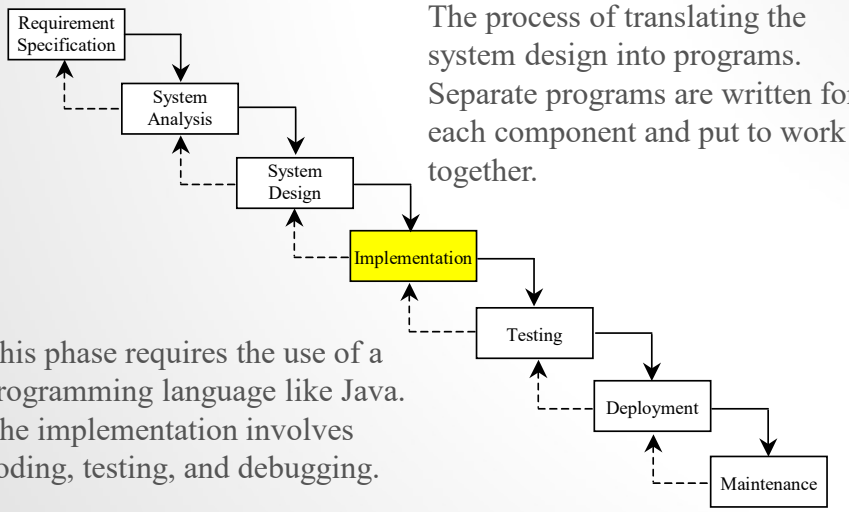
This phase involves the use of many levels of abstraction to decompose the problem into manageable components, identify classes and interfaces, and establish relationships among the classes and interfaces.

IPO



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

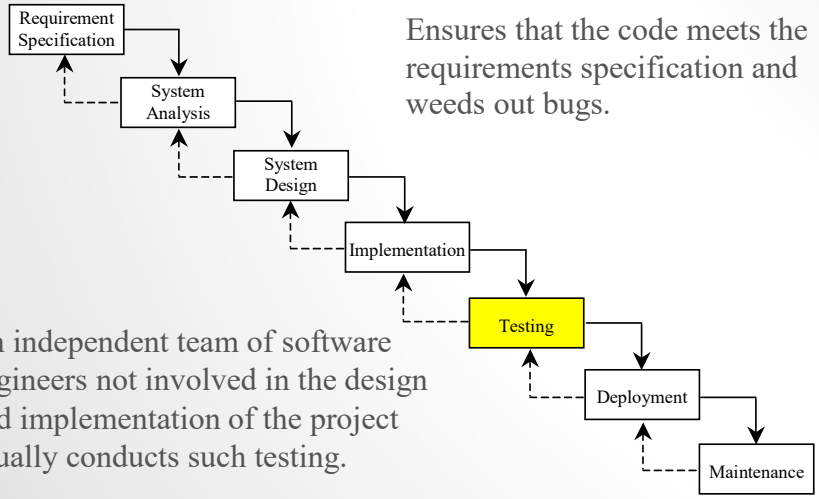
IMPLEMENTATION



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.

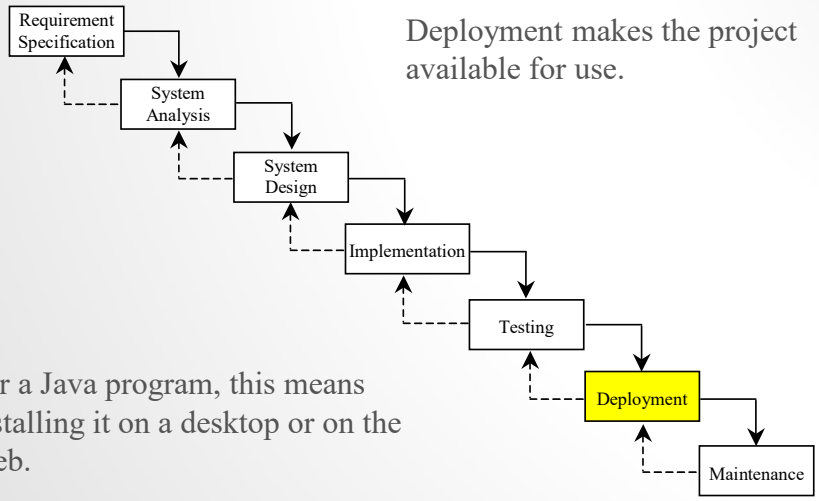
TESTING



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.

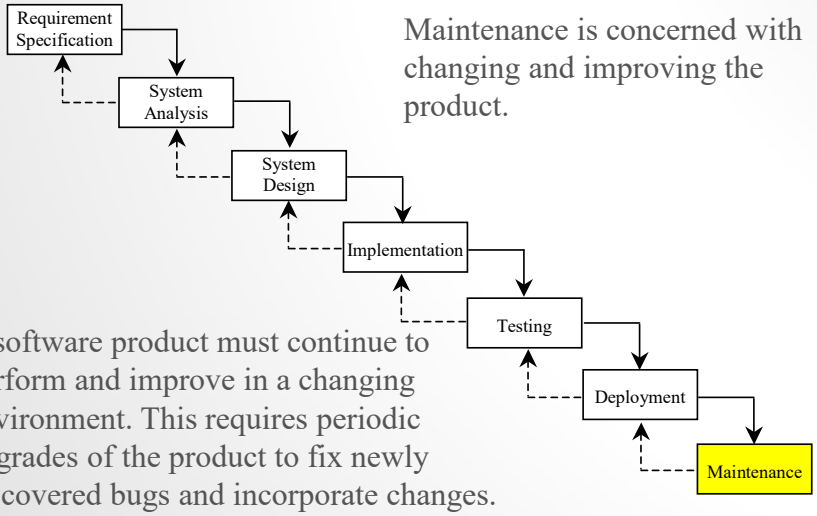
DEPLOYMENT



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 - 0.5**);

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();
```

CSE 101 - COMPUTER PROGRAMMING I SELECTIONS

Joseph LEDET
Department of Computer Engineering
Akdeniz University
josephledet@akdeniz.edu.tr

Motivations

If you assigned a negative value for radius in Listing 2.2, ComputeAreaWithConsoleInput.java, the program would print an invalid result. If the radius is negative, you don't want the program to compute the area. How can you deal with this situation?

Objectives

- To declare **boolean** variables and write Boolean expressions using relational operators (§3.2).
- To implement selection control using one-way **if** statements (§3.3).
- To implement selection control using two-way **if-else** statements (§3.4).
- To implement selection control using nested **if** and multi-way **if** statements (§3.5).
- To avoid common errors and pitfalls in **if** statements (§3.6).
- To generate random numbers using the **Math.random()** method (§3.7).
- To program using selection statements for a variety of examples (**SubtractionQuiz**, **BMI**, **ComputeTax**) (§§3.7–3.9).
- To combine conditions using logical operators (**&&**, ||, and **!**) (§3.10).
- To program using selection statements with combined conditions (**LeapYear**, **Lottery**) (§§3.11–3.12).
- To implement selection control using **switch** statements (§3.13).
- To write expressions using the conditional expression (§3.14).
- To examine the rules governing operator precedence and associativity (§3.15).
- To apply common techniques to debug errors (§3.16).

The `boolean` Type and Operators

Often in a program you need to compare two values, such as whether i is greater than j. Java provides six comparison operators (also known as relational operators) that can be used to compare two values. The result of the comparison is a Boolean value: true or false.

```
boolean b = (1 > 2);
```

Relational Operators

Java Operator	Mathematics Symbol	Name	Example (radius is 5)	Result
<	<	less than	`radius < 0`	`false`
<=	≤	less than or equal to	`radius <= 0`	`false`
>	>	greater than	`radius > 0`	`true`
>=	≥	greater than or equal to	`radius >= 0`	`true`
==	=	equal to	`radius == 0`	`false`
!=	≠	not equal to	`radius != 0`	`true`

Problem: A Simple Math Learning Tool

This example creates a program to let a first grader practice additions. The program randomly generates two single-digit integers number1 and number2 and displays a question such as "What is 7 + 9?" to the student. After the student types the answer, the program displays a message to indicate whether the answer is true or false.

IMPORTANT NOTE: If you cannot run the buttons, see www.cs.armstrong.edu/liang/javaslidenote.doc

.

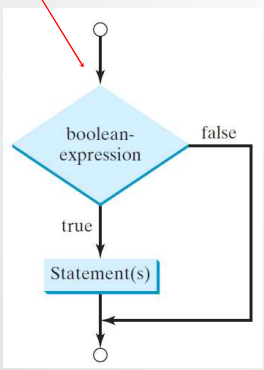
Animation

AdditionQuiz

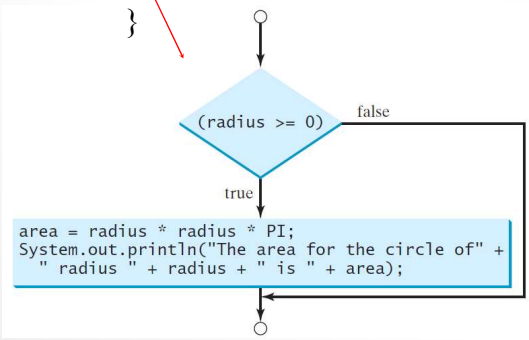
Run

One-way `if` Statements

```
if (boolean-expression) {
  statement(s);
}
```




```
if (radius >= 0) {
  area = radius * radius * PI;
  System.out.println("The area"
    + " for the circle of radius "
    + radius + " is " + area);
}
```



Note

```
if i > 0 {
  System.out.println("i is positive");
}
```

(a) Wrong

```
if (i > 0) {
  System.out.println("i is positive");
}
```

(b) Correct

```
if (i > 0) {
  System.out.println("i is positive");
}
```

(a)

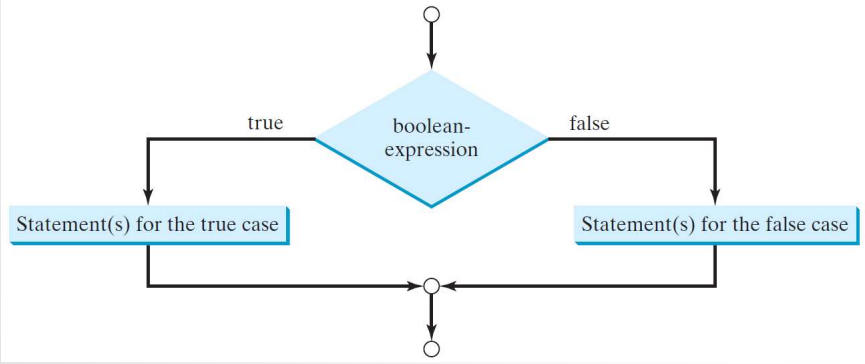
Equivalent

```
if (i > 0)
  System.out.println("i is positive");
```

(b)

The Two-way `if` Statement

```
if (boolean-expression) {
  statement(s)-for-the-true-case;
}
else {
  statement(s)-for-the-false-case;
}
```



`if-else` Example

```
if (radius >= 0) {
  area = radius * radius * 3.14159;

  System.out.println("The area for the "
    + "circle of radius " + radius +
    " is " + area);
}
else {
  System.out.println("Negative input");
}
```

Multiple Alternative if Statements

```
if (score >= 90.0)
  System.out.print("A");
else
  if (score >= 80.0)
    System.out.print("B");
  else
    if (score >= 70.0)
      System.out.print("C");
    else
      if (score >= 60.0)
        System.out.print("D");
      else
        System.out.print("F");
```

(a)

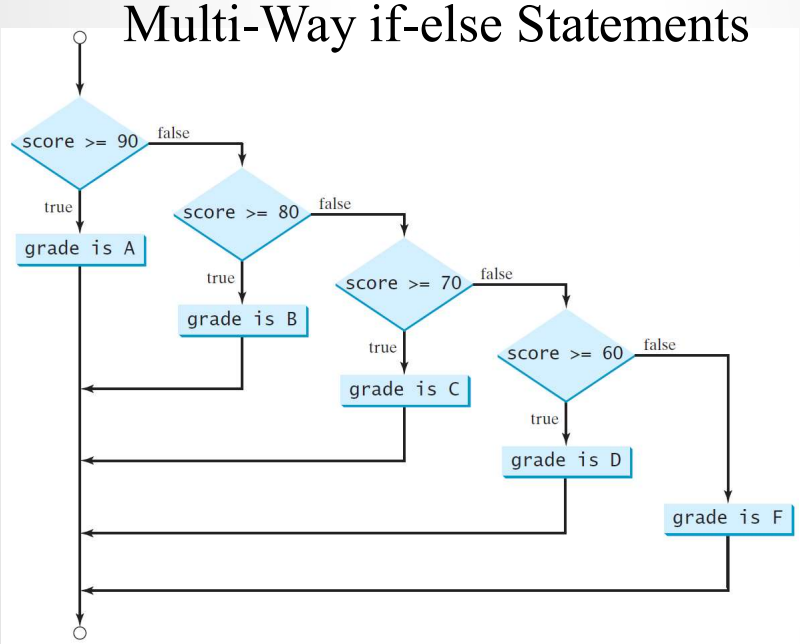
Equivalent

This is better

```
if (score >= 90.0)
  System.out.print("A");
else if (score >= 80.0)
  System.out.print("B");
else if (score >= 70.0)
  System.out.print("C");
else if (score >= 60.0)
  System.out.print("D");
else
  System.out.print("F");
```

(b)

Multi-Way if-else Statements



animation

Trace if-else statement

Suppose score is 70.0

The condition is false

```
if (score >= 90.0)
  System.out.print("A");
else if (score >= 80.0)
  System.out.print("B");
else if (score >= 70.0)
  System.out.print("C");
else if (score >= 60.0)
  System.out.print("D");
else
  System.out.print("F");
```

animation

Trace if-else statement

Suppose score is 70.0

The condition is false

```
if (score >= 90.0)
  System.out.print("A");
else if (score >= 80.0)
  System.out.print("B");
else if (score >= 70.0)
  System.out.print("C");
else if (score >= 60.0)
  System.out.print("D");
else
  System.out.print("F");
```

animation

Trace if-else statement

Suppose score is 70.0

The condition is true

```
if (score >= 90.0)
  System.out.print("A");
else if (score >= 80.0)
  System.out.print("B");
else if (score >= 70.0)
  System.out.print("C");
else if (score >= 60.0)
  System.out.print("D");
else
  System.out.print("F");
```


animation

Trace if-else statement

Suppose score is 70.0

grade is C

```
if (score >= 90.0)
  System.out.print("A");
else if (score >= 80.0)
  System.out.print("B");
else if (score >= 70.0)
  System.out.print("C");
else if (score >= 60.0)
  System.out.print("D");
else
  System.out.print("F");
```


animation

Trace if-else statement

Suppose score is 70.0

Exit the if statement

```
if (score >= 90.0)
  System.out.print("A");
else if (score >= 80.0)
  System.out.print("B");
else if (score >= 70.0)
  System.out.print("C");
else if (score >= 60.0)
  System.out.print("D");
else
  System.out.print("F");
```

Note

The else clause matches the most recent if clause in the same block.

```
int i = 1, j = 2, k = 3;

if (i > j)
  if (i > k)
    System.out.println("A");
else
    System.out.println("B");
```

(a)

Equivalent

This is better with correct indentation

```
int i = 1, j = 2, k = 3;

if (i > j)
  if (i > k)
    System.out.println("A");
  else
    System.out.println("B");
```

(b)

Note, cont.

Nothing is printed from the preceding statement. To force the else clause to match the first if clause, you must add a pair of braces:

```
int i = 1;
int j = 2;
int k = 3;
if (i > j) {
  if (i > k)
    System.out.println("A");
}
else
  System.out.println("B");
```

This statement prints B.

Common Errors

Adding a semicolon at the end of an if clause is a common mistake.

```
if (radius >= 0);
{
 area = radius*radius*PI;
 System.out.println(
   "The area for the circle of radius " +
   radius + " is " + area);
}
```

Wrong

This mistake is hard to find, because it is not a compilation error or a runtime error, it is a logic error.

This error often occurs when you use the next-line block style.

TIP

```
if (number % 2 == 0)
  even = true;
else
  even = false;
```

(a)

Equivalent

```
boolean even
  = number % 2 == 0;
```

(b)

CAUTION

```
if (even == true)
  System.out.println(
    "It is even.");
```

(a)

Equivalent

```
if (even)
  System.out.println(
    "It is even.");
```

(b)

Problem: Body Mass Index

Body Mass Index (BMI) is a measure of health on weight. It can be calculated by taking your weight in kilograms and dividing by the square of your height in meters. The interpretation of BMI for people 16 years or older is as follows:

BMI	Interpretation
BMI < 18.5	Underweight
18.5 <= BMI < 25.0	Normal
25.0 <= BMI < 30.0	Overweight
30.0 <= BMI	Obese

ComputeAndInterpretBMI

Run

Problem: Computing Taxes

The US federal personal income tax is calculated based on the filing status and taxable income. There are four filing statuses: single filers, married filing jointly, married filing separately, and head of household. The tax rates for 2009 are shown below.

Marginal Tax Rate	*Single*	*Married Filing Jointly or Qualifying Widow(er)*	*Married Filing Separately*	*Head of Household*
10%	$0 – $8,350	$0 – $16,700	$0 – $8,350	$0 – $11,950
15%	$8,351 – $33,950	$16,701 – $67,900	$8,351 – $33,950	$11,951 – $45,500
25%	$33,951 – $82,250	$67,901 – $137,050	$33,951 – $68,525	$45,501 – $117,450
28%	$82,251 – $171,550	$137,051 – $208,850	$68,526 – $104,425	$117,451 – $190,200
33%	$171,551 – $372,950	$208,851 – $372,950	$104,426 – $186,475	$190,201 – $372,950
35%	$372,951+	$372,951+	$186,476+	$372,951+

Problem: Computing Taxes, cont.

```
if (status == 0) {
  // Compute tax for single filers
}
else if (status == 1) {
  // Compute tax for married file jointly
  // or qualifying widow(er)
}
else if (status == 2) {
  // Compute tax for married file separately
}
else if (status == 3) {
  // Compute tax for head of household
}
else {
  // Display wrong status
}
```

Logical Operators

Operator	Name	Description
!	not	logical negation
&&	and	logical conjunction
||	or	logical disjunction
^	exclusive or	logical exclusion

Truth Table for Operator !

p	!p	Example (assume age = 24, weight = 140)
true	false	!(age > 18) is false, because (age > 18) is true.
false	true	!(weight == 150) is true, because (weight == 150) is false.

Truth Table for Operator &&

p_1	p_2	p_1 && p_2	Example (assume age = 24, weight = 140)
false	false	false	(age <= 18) && (weight < 140) is false, because (age > 18) and (weight <= 140) are both false.
false	true	false	(age <= 18) && (weight >= 140) is false, because (weight > 140) is false.
true	false	false	(age > 18) && (weight > 140) is false, because (weight > 140) is false.
true	true	true	(age > 18) && (weight >= 140) is true, because both (age > 18) and (weight >= 140) are true.

Truth Table for Operator ||

p_1	p_2	$p_1 \| p_2$	Example (assume age = 24, weight = 140)
false	false	false	(age > 34) || (weight >= 150) is false, because (age > 34) is false and (weight >= 150) is false.
false	true	true	(age > 34) || (weight <= 140) is true, because (age > 34) is false, but (weight <= 140) is true.
true	false	true	(age > 14) || (weight >= 150) is true, because (age > 14) is true.
true	true	true	(age > 14) || (weight <= 140) is true, because (age > 14) is true and (weight <= 140) is true.

Truth Table for Operator ^

p_1	p_2	$p_1 \wedge p_2$	Example (assume age = 24, weight = 140)
false	false	false	(age > 34) ^ (weight > 140) is false, because (age > 34) is false and (weight > 140) is false.
false	true	true	(age > 34) ^ (weight >= 140) is true, because (age > 34) is false but (weight >= 140) is true.
true	false	true	(age > 14) ^ (weight > 140) is true, because (age > 14) is true and (weight > 140) is false.
true	true	false	(age > 14) ^ (weight >= 140) is false, because (age > 14) is true and (weight >= 140) is true.

Examples

Here is a program that checks whether a number is divisible by 2 and 3, whether a number is divisible by 2 or 3, and whether a number is divisible by 2 or 3 but not both:


TestBooleanOperators Run

Examples

```
System.out.println("Is " + number + " divisible by 2 and 3? " +
  ((number % 2 == 0) && (number % 3 == 0)));

System.out.println("Is " + number + " divisible by 2 or 3? " +
  ((number % 2 == 0) || (number % 3 == 0)));

System.out.println("Is " + number +
  " divisible by 2 or 3, but not both? " +
  ((number % 2 == 0) ^ (number % 3 == 0)));
```


TestBooleanOperators Run

Companion Website

The & and | Operators

```
If x is 1, what is x after this
  expression?
(x > 1) & (x++ < 10)

If x is 1, what is x after this
  expression?
(1 > x) && ( 1 > x++)

How about (1 == x) | (10 > x++)?
(1 == x) || (10 > x++)?
```

Problem: Determining Leap Year?

This program first prompts the user to enter a year as an int value and checks if it is a leap year.

A year is a leap year if it is divisible by 4 but not by 100, or it is divisible by 400.

(year % 4 == 0 && year % 100 != 0) || (year % 400 == 0)

LeapYear Run

Problem: Lottery

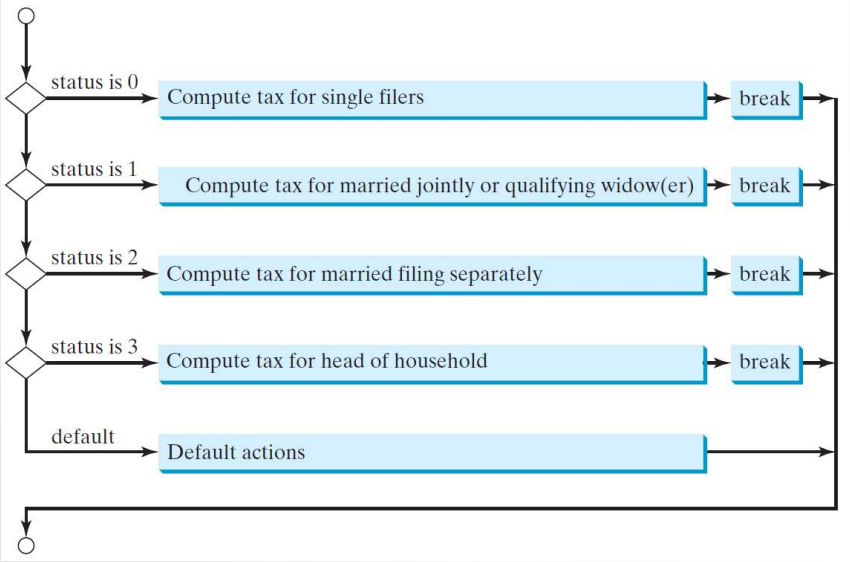
Write a program that randomly generates a lottery of a two-digit number, prompts the user to enter a two-digit number, and determines whether the user wins according to the following rule:

- If the user input matches the lottery in exact order, the award is $10,000.
- If the user input matches the lottery, the award is $3,000.
- If one digit in the user input matches a digit in the lottery, the award is $1,000.

`switch` Statements

```
switch (status) {
  case 0:  compute taxes for single filers;
           break;
  case 1:  compute taxes for married file jointly;
           break;
  case 2:  compute taxes for married file separately;
           break;
  case 3:  compute taxes for head of household;
           break;
  default: System.out.println("Errors: invalid status");
           System.exit(1);
}
```

`switch` Statement Flow Chart



`switch` Statement Rules

The switch-expression must yield a value of char, byte, short, or int type and must always be enclosed in parentheses.

The value1, ..., and valueN must have the same data type as the value of the switch-expression. The resulting statements in the case statement are executed when the value in the case statement matches the value of the switch-expression. Note that value1, ..., and valueN are constant expressions, meaning that they cannot contain variables in the expression, such as 1 + x.

```
switch (switch-expression) {
  case value1:  statement(s)1;
                break;
  case value2: statement(s)2;
                break;
  …
  case valueN: statement(s)N;
                break;
  default: statement(s)-for-default;
}
```

`switch` Statement Rules

The keyword break is optional, but it should be used at the end of each case in order to terminate the remainder of the switch statement. If the break statement is not present, the next case statement will be executed.

The default case, which is optional, can be used to perform actions when none of the specified cases matches the switch-expression.

```
switch (switch-expression) {
  case value1:  statement(s)1;
        break;
  case value2: statement(s)2;
        break;
  …
  case valueN: statement(s)N;
        break;
  default: statement(s)-for-default;
}
```

When the value in a **case** statement matches the value of the **switch-expression**, the statements *starting from this case* are executed until either a **break** statement or the end of the **switch** statement is reached.

animation

Trace switch statement

Suppose day is 2:

```
switch (day) {
  case 1:
  case 2:
  case 3:
  case 4:
  case 5: System.out.println("Weekday"); break;
  case 0:
  case 6: System.out.println("Weekend");
}
```


animation

Trace switch statement

Match case 2

```
switch (day) {
  case 1:
  case 2:
  case 3:
  case 4:
  case 5: System.out.println("Weekday"); break;
  case 0:
  case 6: System.out.println("Weekend");
}
```


animation

Trace switch statement

Fall through case 3

```
switch (day) {
  case 1:
  case 2:
  case 3:
  case 4:
  case 5: System.out.println("Weekday"); break;
  case 0:
  case 6: System.out.println("Weekend");
}
```


animation

Trace switch statement

Fall through case 4

```
switch (day) {
  case 1:
  case 2:
  case 3:
  case 4:
  case 5: System.out.println("Weekday"); break;
  case 0:
  case 6: System.out.println("Weekend");
}
```

animation

Trace switch statement

Fall through case 5

```
switch (day) {
  case 1:
  case 2:
  case 3:
  case 4:
  case 5: System.out.println("Weekday"); break;
  case 0:
  case 6: System.out.println("Weekend");
}
```

animation

Trace switch statement

Encounter break

```
switch (day) {
  case 1:
  case 2:
  case 3:
  case 4:
  case 5: System.out.println("Weekday"); break;
  case 0:
  case 6: System.out.println("Weekend");
}
```


animation

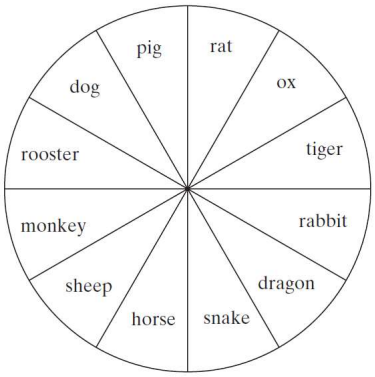
Trace switch statement

Exit the statement

```
swit  day) {
  ca  :
  ca  2:
  c  3:
  c  e 4:
  c  se 5: System.out.println("Weekday"); break;
  ase 0:
  ase 6: System.out.println("Weekend");
}
```

Problem: Chinese Zodiac

Write a program that prompts the user to enter a year and displays the animal for the year.



year % 12 =
0: monkey
1: rooster
2: dog
3: pig
4: rat
5: ox
6: tiger
7: rabbit
8: dragon
9: snake
10: horse
11: sheep

ChineseZodiac

Run

Conditional Expressions

```
if (x > 0)
  y = 1
else
  y = -1;
```

is equivalent to

```
y = (x > 0) ? 1 : -1;
```

(boolean-expression) ? expression1 : expression2

Ternary operator
Binary operator
Unary operator

Conditional Operator

```
if (num % 2 == 0)
  System.out.println(num + "is even");
else
  System.out.println(num + "is odd");


System.out.println(
  (num % 2 == 0)? num + "is even" :
  num + "is odd");
```


Conditional Operator, cont.

```
boolean-expression ? exp1 : exp2
```


Operator Precedence

- var++, var--
- +, - (Unary plus and minus), ++var,--var
- (type) Casting
- ! (Not)
- *, /, % (Multiplication, division, and remainder)
- +, - (Binary addition and subtraction)
- <, <=, >, >= (Relational operators)
- ==, !=; (Equality)
- ^ (Exclusive OR)
- && (Conditional AND) Short-circuit AND
- || (Conditional OR) Short-circuit OR
- =, +=, -=, *=, /=, %= (Assignment operator)

Operator Precedence and Associativity

The expression in the parentheses is evaluated first. (Parentheses can be nested, in which case the expression in the inner parentheses is executed first.) When evaluating an expression without parentheses, the operators are applied according to the precedence rule and the associativity rule.

If operators with the same precedence are next to each other, their associativity determines the order of evaluation. All binary operators except assignment operators are left-associative.

Operator Associativity

When two operators with the same precedence are evaluated, the *associativity* of the operators determines the order of evaluation. All binary operators except assignment operators are *left-associative*.

a – b + c – d is equivalent to ((a – b) + c) – d

Assignment operators are *right-associative*. Therefore, the expression

a = b += c = 5 is equivalent to a = (b += (c = 5))

Example

Applying the operator precedence and associativity rule, the expression 3 + 4 * 4 > 5 * (4 + 3) - 1 is evaluated as follows:

```
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) subtraction

```
19 > 34
```
(6) greater than

```
false
```

Companion Website

Operand Evaluation Order

Supplement III.A, "Advanced discussions on how an expression is evaluated in the JVM."

Debugging

Logic errors are called *bugs*. The process of finding and correcting errors is called debugging. A common approach to debugging is to use a combination of methods to narrow down to the part of the program where the bug is located. You can hand-trace the program (i.e., catch errors by reading the program), or you can insert print statements in order to show the values of the variables or the execution flow of the program. This approach might work for a short, simple program. But for a large, complex program, the most effective approach for debugging is to use a debugger utility.

Debugger

Debugger is a program that facilitates debugging.
You can use a debugger to

- Execute a single statement at a time.
- Trace into or stepping over a method.
- Set breakpoints.
- Display variables.
- Display call stack.
- Modify variables.

CSE 101 - COMPUTER PROGRAMMING I MATHEMATICAL FUNCTIONS, CHARACTERS, AND STRINGS

Joseph LEDET
Department of Computer Engineering
Akdeniz University
josephledet@akdeniz.edu.tr

MATHEMATICAL FUNCTIONS

Java provides many useful methods in the **Math** class for performing common mathematical functions.

THE MATH CLASS

- Class constants:
  - PI
  - E
- Class methods:
  - Trigonometric Methods
  - Exponent Methods
  - Rounding Methods
  - min, max, abs, and random Methods

TRIGONOMETRIC METHODS

- **sin(double a)**
- **cos(double a)**
- **tan(double a)**
- **acos(double a)**
- **asin(double a)**
- **atan(double a)**

Radians

toRadians(90)

```
Examples:

Math.sin(0) returns 0.0
Math.sin(Math.PI / 6)
  returns 0.5
Math.sin(Math.PI / 2)
  returns 1.0
Math.cos(0) returns 1.0
Math.cos(Math.PI / 6)
  returns 0.866
Math.cos(Math.PI / 2)
  returns 0
```

EXPONENT METHODS

- **`exp(double a)`**
  Returns `e` raised to the power of `a`.
- **`log(double a)`**
  Returns the natural logarithm of `a`.
- **`log10(double a)`**
  Returns the 10-based logarithm of `a`.
- **`pow(double a, double b)`**
  Returns `a` raised to the power of `b`.
- **`sqrt(double a)`**
  Returns the square root of `a`.

Examples:

```
Math.exp(1) returns 2.71
Math.log(2.71) returns 1.0
Math.pow(2, 3) returns 8.0
Math.pow(3, 2) returns 9.0
Math.pow(3.5, 2.5) returns
   22.91765
Math.sqrt(4) returns 2.0
Math.sqrt(10.5) returns 3.24
```

ROUNDING METHODS

- **`double ceil(double x)`**
  x rounded up to its nearest integer. This integer is returned as a double value.
- **`double floor(double x)`**
  x is rounded down to its nearest integer. This integer is returned as a double value.
- **`double rint(double x)`**
  x is rounded to its nearest integer. If x is equally close to two integers, the even one is returned as a double.
- **`int round(float x)`**
  Return (int)Math.floor(x+0.5).
- **`long round(double x)`**
  Return (long)Math.floor(x+0.5).

ROUNDING METHODS EXAMPLES

```
Math.ceil(2.1) returns 3.0
Math.ceil(2.0) returns 2.0
Math.ceil(-2.0) returns -2.0
Math.ceil(-2.1) returns -2.0
Math.floor(2.1) returns 2.0
Math.floor(2.0) returns 2.0
Math.floor(-2.0) returns -2.0
Math.floor(-2.1) returns -3.0
Math.rint(2.1) returns 2.0
Math.rint(2.0) returns 2.0
Math.rint(-2.0) returns -2.0
Math.rint(-2.1) returns -2.0
Math.rint(2.5) returns 2.0
Math.rint(-2.5) returns -2.0
Math.round(2.6f) returns 3
Math.round(2.0) returns 2
Math.round(-2.0f) returns -2
Math.round(-2.6) returns -3
```

MIN, MAX, AND ABS

- `max(a, b)` and `min(a, b)`
  Returns the maximum or minimum of two parameters.
- `abs(a)`
  Returns the absolute value of the parameter.
- `random()`
  Returns a random `double` value in the range [0.0, 1.0).

Examples:

```
Math.max(2, 3) returns 3
Math.max(2.5, 3) returns
  3.0
Math.min(2.5, 3.6)
  returns 2.5
Math.abs(-2) returns 2
Math.abs(-2.1) returns
  2.1
```

THE RANDOM METHOD

Generates a random double value greater than or equal to 0.0 and less than 1.0 (0 <= Math.random() < 1.0).

Examples:

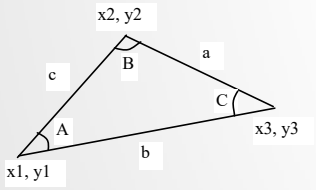
```
(int)(Math.random() * 10)
```
→ Returns a random integer between 0 and 9.

```
50 + (int)(Math.random() * 50)
```
→ Returns a random integer between 50 and 99.

In general,

```
a + Math.random() * b
```
→ Returns a random number between a and a + b, excluding a + b.

CASE STUDY: COMPUTING ANGLES OF A TRIANGLE




```
A = acos((a * a - b * b - c * c) / (-2 * b * c))
B = acos((b * b - a * a - c * c) / (-2 * a * c))
C = acos((c * c - b * b - a * a) / (-2 * a * b))
```

Write a program that prompts the user to enter the x- and y-coordinates of the three corner points in a triangle and then displays the triangle's angles.

IMPORTANT NOTE: If you cannot run the buttons, see www.cs.armstrong.edu/liang/javaslidenote.doc

ComputeAngles

Run

CHARACTER DATA TYPE

```
char letter = 'A'; (ASCII)
char numChar = '4'; (ASCII)
char letter = '\u0041'; (Unicode)
char numChar = '\u0034'; (Unicode)
```

Four hexadecimal digits.

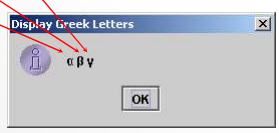
NOTE: The increment and decrement operators can also be used on char variables to get the next or preceding Unicode character. For example, the following statements display character b.

```
char ch = 'a';
System.out.println(++ch);
```


UNICODE FORMAT

Java characters use *Unicode*, a 16-bit encoding scheme established by the Unicode Consortium to support the interchange, processing, and display of written texts in the world's diverse languages. Unicode takes two bytes, preceded by \u, expressed in four hexadecimal numbers that run from '\u0000' to '\uFFFF'. So, Unicode can represent `65535 + 1 characters`.

Unicode \u03b1 \u03b2 \u03b3 for three Greek letters



ASCII CODE FOR COMMONLY USED CHARACTERS

Characters	Code Value in Decimal	Unicode Value
'0' to '9'	48 to 57	\u0030 to \u0039
'A' to 'Z'	65 to 90	\u0041 to \u005A
'a' to 'z'	97 to 122	\u0061 to \u007A

ESCAPE SEQUENCES FOR SPECIAL CHARACTERS

Escape Sequence	*Name*	*Unicode Code*	*Decimal Value*
\b	Backspace	\u0008	8
\t	Tab	\u0009	9
\n	Linefeed	\u000A	10
\f	Formfeed	\u000C	12
\r	Carriage Return	\u000D	13
\\	Backslash	\u005C	92
\"	Double Quote	\u0022	34

APPENDIX B: ASCII CHARACTER SET

ASCII Character Set is a subset of the Unicode from \u0000 to \u007f

TABLE B.1 ASCII Character Set in the Decimal Index

	0	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*
0	nul	soh	stx	etx	eot	enq	ack	bel	bs	ht
1	nl	vt	ff	cr	so	si	dle	dcl	dc2	dc3
2	dc4	nak	syn	etb	can	em	sub	esc	fs	gs
3	rs	us	sp	!	"	#	$	%	&	'
4	(	)	*	+	,	-	.	/	0	1
5	2	3	4	5	6	7	8	9	:	;
6	<	=	>	?	@	A	B	C	D	E
7	F	G	H	I	J	K	L	M	N	O
8	P	Q	R	S	T	U	V	W	X	Y
9	Z	[	\	]	^	_	`	a	b	c
10	d	e	f	g	h	i	j	k	l	m
11	n	o	p	q	r	s	t	u	v	w
12	x	y	z	{	|	}	~	del		

ASCII CHARACTER SET, CONT.

ASCII Character Set is a subset of the Unicode from \u0000 to \u007f

TABLE B.2 ASCII Character Set in the Hexadecimal Index

	0	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*A*	*B*	*C*	*D*	*E*	*F*
0	nul	soh	stx	etx	eot	enq	ack	bel	bs	ht	nl	vt	ff	cr	so	si
1	dle	dcl	dc2	dc3	dc4	nak	syn	etb	can	em	sub	esc	fs	gs	rs	us
2	sp	!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	del

CASTING BETWEEN CHAR AND NUMERIC TYPES

```
int i = 'a'; // Same as int i = (int)'a';

char c = 97; // Same as char c = (char)97;
```

COMPARING AND TESTING CHARACTERS

```
if (ch >= 'A' && ch <= 'Z')
  System.out.println(ch + " is an uppercase letter");
else if (ch >= 'a' && ch <= 'z')
  System.out.println(ch + " is a lowercase letter");
else if (ch >= '0' && ch <= '9')
  System.out.println(ch + " is a numeric character");
```

METHODS IN THE CHARACTER CLASS

Method	Description
`isDigit(ch)`	Returns true if the specified character is a digit.
`isLetter(ch)`	Returns true if the specified character is a letter.
`isLetterOrDigit(ch)`	Returns true if the specified character is a letter or digit.
`isLowerCase(ch)`	Returns true if the specified character is a lowercase letter.
`isUpperCase(ch)`	Returns true if the specified character is an uppercase letter.
`toLowerCase(ch)`	Returns the lowercase of the specified character.
`toUpperCase(ch)`	Returns the uppercase of the specified character.

THE STRING TYPE

The char type only represents one character. To represent a string of characters, use the data type called String. For example,

String message = "Welcome to Java";

String is actually a predefined class in the Java library just like the System class and Scanner class. The String type is not a primitive type. It is known as a *reference type*. Any Java class can be used as a reference type for a variable. Reference data types will be thoroughly discussed in Chapter 9, "Objects and Classes." For the time being, you just need to know how to declare a String variable, how to assign a string to the variable, how to concatenate strings, and to perform simple operations for strings.

SIMPLE METHODS FOR **STRING** OBJECTS

Method	Description
`length()`	Returns the number of characters in this string.
`charAt(index)`	Returns the character at the specified index from this string.
`concat(s1)`	Returns a new string that concatenates this string with string s1.
`toUpperCase()`	Returns a new string with all letters in uppercase.
`toLowerCase()`	Returns a new string with all letters in lowercase.
`trim()`	Returns a new string with whitespace characters trimmed on both sides.

SIMPLE METHODS FOR **STRING** OBJECTS

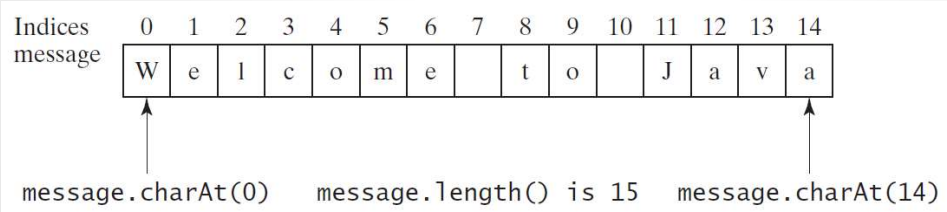
Strings are objects in Java. The methods in the preceding table can only be invoked from a specific string instance. For this reason, these methods are called *instance methods*. A non-instance method is called a *static method*. A static method can be invoked without using an object. All the methods defined in the **Math** class are static methods. They are not tied to a specific object instance. The syntax to invoke an instance method is

referenceVariable.methodName(arguments).

GETTING STRING LENGTH

```
String message = "Welcome to Java";
System.out.println("The length of " + message + " is "
  + message.length());
```


GETTING CHARACTERS FROM A STRING




```
String message = "Welcome to Java";
System.out.println("The first character in message is "
  + message.charAt(0));
```

CONVERTING STRINGS

"Welcome".toLowerCase() returns a new string, welcome.

"Welcome".toUpperCase() returns a new string, WELCOME.

" Welcome ".trim() returns a new string, Welcome.

STRING CONCATENATION

String s3 = s1.concat(s2); or String s3 = s1 + s2;

```
// Three strings are concatenated
String message = "Welcome " + "to " + "Java";

// String Chapter is concatenated with number 2
String s = "Chapter" + 2; // s becomes Chapter2

// String Supplement is concatenated with
character B
String s1 = "Supplement" + 'B'; // s1 becomes
SupplementB
```

READING A STRING FROM THE CONSOLE

```
Scanner input = new Scanner(System.in);
System.out.print("Enter three words separated by spaces: ");
String s1 = input.next();
String s2 = input.next();
String s3 = input.next();
System.out.println("s1 is " + s1);
System.out.println("s2 is " + s2);
System.out.println("s3 is " + s3);
```


READING A CHARACTER FROM THE CONSOLE

```
Scanner input = new Scanner(System.in);
System.out.print("Enter a character: ");
String s = input.nextLine();
char ch = s.charAt(0);
System.out.println("The character entered is " + ch);
```


COMPARING STRINGS

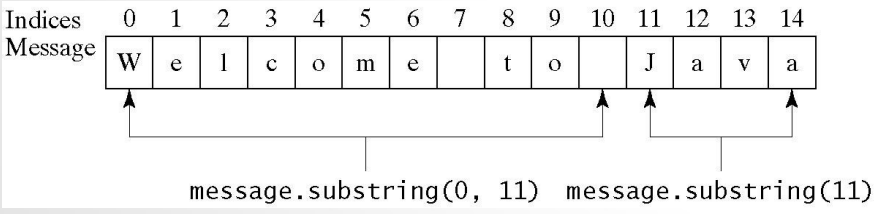
Method	Description
`equals(s1)`	Returns true if this string is equal to string `s1`.
`equalsIgnoreCase(s1)`	Returns true if this string is equal to string `s1`; it is case insensitive.
`compareTo(s1)`	Returns an integer greater than `0`, equal to `0`, or less than `0` to indicate whether this string is greater than, equal to, or less than `s1`.
`compareToIgnoreCase(s1)`	Same as `compareTo` except that the comparison is case insensitive.
`startsWith(prefix)`	Returns true if this string starts with the specified prefix.
`endsWith(suffix)`	Returns true if this string ends with the specified suffix.

OrderTwoCities

Run

OBTAINING SUBSTRINGS

Method	Description
`substring(beginIndex)`	Returns this string's substring that begins with the character at the specified `beginIndex` and extends to the end of the string, as shown in Figure 4.2.
`substring(beginIndex, endIndex)`	Returns this string's substring that begins at the specified `beginIndex` and extends to the character at index `endIndex - 1`, as shown in Figure 9.6. Note that the character at `endIndex` is not part of the substring.

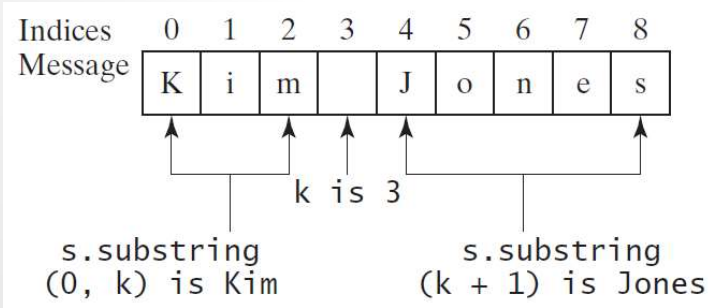


FINDING A CHARACTER OR A SUBSTRING IN A STRING

Method	Description
`indexOf(ch)`	Returns the index of the first occurrence of `ch` in the string. Returns `-1` if not matched.
`indexOf(ch, fromIndex)`	Returns the index of the first occurrence of `ch` after `fromIndex` in the string. Returns `-1` if not matched.
`indexOf(s)`	Returns the index of the first occurrence of string `s` in this string. Returns `-1` if not matched.
`indexOf(s, fromIndex)`	Returns the index of the first occurrence of string `s` in this string after `fromIndex`. Returns `-1` if not matched.
`lastIndexOf(ch)`	Returns the index of the last occurrence of `ch` in the string. Returns `-1` if not matched.
`lastIndexOf(ch, fromIndex)`	Returns the index of the last occurrence of `ch` before `fromIndex` in this string. Returns `-1` if not matched.
`lastIndexOf(s)`	Returns the index of the last occurrence of string `s`. Returns `-1` if not matched.
`lastIndexOf(s, fromIndex)`	Returns the index of the last occurrence of string `s` before `fromIndex`. Returns `-1` if not matched.

FINDING A CHARACTER OR A SUBSTRING IN A STRING

```
int k = s.indexOf(' ');
String firstName = s.substring(0, k);
String lastName = s.substring(k + 1);
```

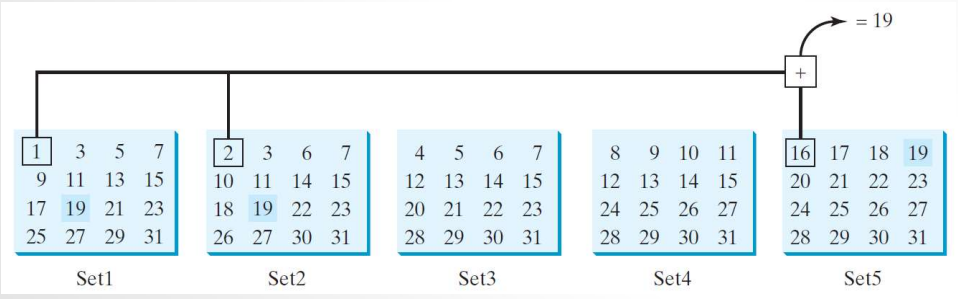


CONVERSION BETWEEN STRINGS AND NUMBERS

```
int intValue = Integer.parseInt(intString);
double doubleValue = Double.parseDouble(doubleString);

String s = number + "";
```

PROBLEM: GUESSING BIRTHDAY

The program can guess your birth date. Run to see how it works.



GuessBirthday

Run

MATHEMATICS BASIS FOR THE GAME

19 is 10011 in binary. 7 is 111 in binary. 23 is 11101 in binary

```
  10000
     10
+     1
  10011
```

19

```
  00110
     10
+     1
  00111
```

7

```
  10000
   1000
    100
+     1
  11101
```

23

Decimal	Binary
1	00001
2	00010
3	00011
...	
19	10011
...	
31	11111

$$
\begin{array}{r}
b_5\ 0\ 0\ 0\ 0 \\
b_4\ 0\ 0\ 0 \\
b_3\ 0\ 0 \\
b_2\ 0 \\
+\qquad b_1 \\
\hline
b_5\, b_4\, b_3\, b_2\, b_1
\end{array}
$$

$$
\begin{array}{r}
10000 \\
10 \\
+\quad 1 \\
\hline
10011
\end{array}
$$

19

$$
\begin{array}{r}
10000 \\
1000 \\
100 \\
10 \\
+\quad 1 \\
\hline
11111
\end{array}
$$

31

CASE STUDY: CONVERTING A HEXADECIMAL DIGIT TO A DECIMAL VALUE

Write a program that converts a hexadecimal digit into a decimal value.

HexDigit2Dec

Run

CASE STUDY: REVISING THE LOTTERY PROGRAM USING STRINGS

A problem can be solved using many different approaches. This section rewrites the lottery program in Listing 3.7 using strings. Using strings simplifies this program.

LotteryUsingStrings

Run

FORMATTING OUTPUT

Use the printf statement.

System.out.printf(format, items);

Where format is a string that may consist of substrings and format specifiers. A format specifier specifies how an item should be displayed. An item may be a numeric value, character, boolean value, or a string. Each specifier begins with a percent sign.

FREQUENTLY-USED SPECIFIERS

Specifier	Output	Example
%b	a boolean value	true or false
%c	a character	'a'
%d	a decimal integer	200
%f	a floating-point number	45.460000
%e	a number in standard scientific notation	4.556000e+01
%s	a string	"Java is cool"

```
int count = 5;
double amount = 45.56;
System.out.printf("count is %d and amount is %f", count, amount);

display            count is 5 and amount is 45.560000
```

items

FORMATDEMO

The example gives a program that uses **printf** to display a table.

FormatDemo Run

CSE 101 - COMPUTER PROGRAMMING I REPETITION

Joseph LEDET
Department of Computer Engineering
Akdeniz University
josephledet@akdeniz.edu.tr

REPETITION

- Sometimes, we want programs to do the same thing multiple times
  - Similar to a copy-paste in a text document
- Sometimes, we want programs to do the same thing multiple times
  - Similar to a copy-paste in a text document
- Sometimes, we want programs to do the same thing multiple times
  - Similar to a copy-paste in a text document

MOTIVATIONS

Suppose that you need to print a string (e.g., "Welcome to Java!") a hundred times. It would be tedious to have to write the following statement a hundred times:

```
System.out.println("Welcome to Java!");
```

So, how do you solve this problem?

OPENING PROBLEM

Problem:

```
                 System.out.println("Welcome to Java!");
                 System.out.println("Welcome to Java!");
                 System.out.println("Welcome to Java!");
                 System.out.println("Welcome to Java!");
                 System.out.println("Welcome to Java!");
                 System.out.println("Welcome to Java!");
100
times
                 ...
                 ...
                 ...
                 System.out.println("Welcome to Java!");
                 System.out.println("Welcome to Java!");
                 System.out.println("Welcome to Java!");
```

INTRODUCING WHILE LOOPS

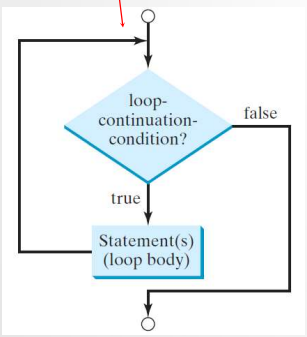
```
int count = 0;
while (count < 100) {
  System.out.println("Welcome to Java");
  count++;
}
```

OBJECTIVES

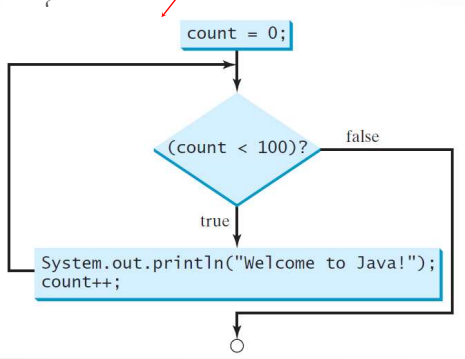
- To write programs for executing statements repeatedly using a **while** loop (§5.2).
- To follow the loop design strategy to develop loops (§§5.2.1–5.2.3).
- To control a loop with a sentinel value (§5.2.4).
- To obtain large input from a file using input redirection rather than typing from the keyboard (§5.2.5).
- To write loops using **do-while** statements (§5.3).
- To write loops using **for** statements (§5.4).
- To discover the similarities and differences of three types of loop statements (§5.5).
- To write nested loops (§5.6).
- To learn the techniques for minimizing numerical errors (§5.7).
- To learn loops from a variety of examples (**GCD**, **FutureTuition**, **Dec2Hex**) (§5.8).
- To implement program control with **break** and **continue** (§5.9).
- To write a program that displays prime numbers (§5.11).

WHILE LOOP FLOW CHART

```
while (loop-continuation-condition) {
  // loop-body;
  Statement(s);
}
```




```
int count = 0;
while (count < 100) {
  System.out.println("Welcome to Java!");
  count++;
}
```



animation

TRACE WHILE LOOP

```
int count = 0;
while (count < 2) {
  System.out.println("Welcome to Java!");
  count++;
}
```

Initialize count

animation

TRACE WHILE LOOP, CONT.

```
int count = 0;
while (count < 2) {
  System.out.println("Welcome to Java!");
  count++;
}
```

(count < 2) is true

animation

TRACE WHILE LOOP, CONT.

```
int count = 0;
while (count < 2) {
  System.out.println("Welcome to Java!");
  count++;
}
```

Print Welcome to Java

animation

TRACE WHILE LOOP, CONT.

```
int count = 0;
while (count < 2) {
  System.out.println("Welcome to Java!");
  count++;
}
```

Increase count by 1
count is 1 now

animation

TRACE WHILE LOOP, CONT.

```
int count = 0;
while (count < 2) {
  System.out.println("Welcome to Java!");
  count++;
}
```

(count < 2) is still true since count is 1

animation

TRACE WHILE LOOP, CONT.

```
int count = 0;
while (count < 2) {
  System.out.println("Welcome to Java!");
  count++;
}
```

Print Welcome to Java

animation

TRACE WHILE LOOP, CONT.

```
int count = 0;
while (count < 2) {
  System.out.println("Welcome to Java!");
  count++;
}
```

Increase count by 1
count is 2 now

animation

TRACE WHILE LOOP, CONT.

```
int count = 0;
while (count < 2) {
  System.out.println("Welcome to Java!");
  count++;
}
```

(count < 2) is false since count is 2 now

animation

TRACE WHILE LOOP

```
int count = 0;
while (count < 2) {
  System.out.println("Welcome to Java!");
  count++;
}
```

The loop exits. Execute the next statement after the loop.

PROBLEM: REPEAT ADDITION UNTIL CORRECT

Recall that Listing 3.1 AdditionQuiz.java gives a program that prompts the user to enter an answer for a question on addition of two single digits. Using a loop, you can now rewrite the program to let the user enter a new answer until it is correct.

IMPORTANT NOTE: If you cannot run the buttons, see www.cs.armstrong.edu/liang/javaslidenote.doc

RepeatAdditionQuiz Run

PROBLEM: GUESSING NUMBERS

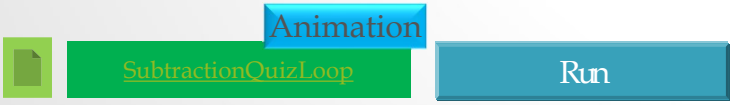
Write a program that randomly generates an integer between 0 and 100, inclusive. The program prompts the user to enter a number continuously until the number matches the randomly generated number. For each user input, the program tells the user whether the input is too low or too high, so the user can choose the next input intelligently. Here is a sample run:

GuessNumberOneTime Run

GuessNumber Run

PROBLEM: AN ADVANCED MATH LEARNING TOOL

The Math subtraction learning tool program generates just one question for each run. You can use a loop to generate questions repeatedly. This example gives a program that generates five questions and reports the number of the correct answers after a student answers all five questions.



ENDING A LOOP WITH A SENTINEL VALUE

Often the number of times a loop is executed is not predetermined. You may use an input value to signify the end of the loop. Such a value is known as a *sentinel value*.

Write a program that reads and calculates the sum of an unspecified number of integers. The input 0 signifies the end of the input.

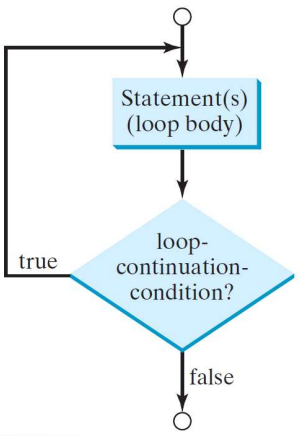
CAUTION

Don't use floating-point values for equality checking in a loop control. Since floating-point values are approximations for some values, using them could result in imprecise counter values and inaccurate results. Consider the following code for computing 1 + 0.9 + 0.8 + ... + 0.1:

```
double item = 1; double sum = 0;
while (item != 0) { // No guarantee item will be 0
  sum += item;
  item -= 0.1;
}
System.out.println(sum);
```


DO-WHILE LOOP

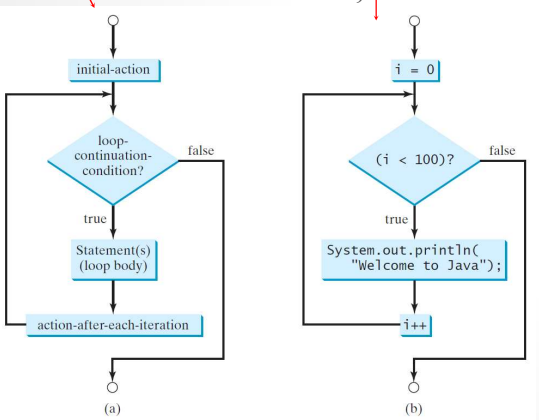
```
do {
  // Loop body;
  Statement(s);
} while (loop-continuation-condition);
```



FOR LOOPS

```
for (initial-action; loop-
   continuation-condition;
   action-after-each-iteration) {
   // loop body;
   Statement(s);
}
```

```
int i;
for (i = 0; i < 100; i++) {
  System.out.println(
    "Welcome to Java!");
}
```



animation

TRACE FOR LOOP

```
int i;
for (i = 0; i < 2; i++) {
  System.out.println(
    "Welcome to Java!");
}
```

Declare i

animation

TRACE FOR LOOP, CONT.

```
int i;
for (i = 0; i < 2; i++) {
  System.out.println(
    "Welcome to Java!");
}
```

Execute initializer
i is now 0

animation

TRACE FOR LOOP, CONT.

```
int i;
for (i = 0; i < 2; i++) {
  System.out.println( "Welcome to Java!");
}
```

(i < 2) is true
since i is 0

animation

TRACE FOR LOOP, CONT.

```
int i;
for (i = 0; i < 2; i++) {
  System.out.println("Welcome to Java!");
}
```

Print Welcome to Java

animation

TRACE FOR LOOP, CONT.

```
int i;
for (i = 0; i < 2; i++) {
  System.out.println("Welcome to Java!");
}
```

Execute adjustment statement
i now is 1

animation

TRACE FOR LOOP, CONT.

```
int i;
for (i = 0; i < 2; i++) {
  System.out.println("Welcome to Java!");
}
```

(i < 2) is still true since i is 1

animation

TRACE FOR LOOP, CONT.

```
int i;
for (i = 0; i < 2; i++) {
  System.out.println("Welcome to Java!");
}
```

Print Welcome to Java

animation

TRACE FOR LOOP, CONT.

```
int i;
for (i = 0; i < 2; i++) {
  System.out.println("Welcome to Java!");
}
```

Execute adjustment statement
i now is 2

animation

TRACE FOR LOOP, CONT.

```
int i;
for (i = 0; i < 2; i++) {
  System.out.println("Welcome to Java!");
}
```

(i < 2) is false
since i is 2

animation

TRACE FOR LOOP, CONT.

```
int i;
for (i = 0; i < 2; i++) {
  System.out.println("Welcome to Java!");
}
```

Exit the loop. Execute the next statement after the loop

NOTE

The initial-action in a for loop can be a list of zero or more comma-separated expressions. The action-after-each-iteration in a for loop can be a list of zero or more comma-separated statements. Therefore, the following two for loops are correct. They are rarely used in practice, however.

```
for (int i = 1; i < 100; System.out.println(i++));


for (int i = 0, j = 0; (i + j < 10); i++, j++) {
  // Do something
}
```

NOTE

If the loop-continuation-condition in a for loop is omitted, it is implicitly true. Thus the statement given below in (a), which is an infinite loop, is correct. Nevertheless, it is better to use the equivalent loop in (b) to avoid confusion:

```
for ( ; ; ) {
  // Do something
}
```

(a)

Equivalent

```
while (true) {
  // Do something
}
```

(b)

CAUTION

Adding a semicolon at the end of the for clause before the loop body is a common mistake, as shown below:

Logic Error

```
for (int i=0; i<10; i++);
{
  System.out.println("i is " + i);
}
```

CAUTION, CONT.

Similarly, the following loop is also wrong:

```
int i=0;
while (i < 10);
{
  System.out.println("i is " + i);
  i++;
}
```

Logic Error

In the case of the do loop, the following semicolon is needed to end the loop.

```
int i=0;
do {
  System.out.println("i is " + i);
  i++;
} while (i<10);
```

Correct

WHICH LOOP TO USE?

The three forms of loop statements, while, do-while, and for, are expressively equivalent; that is, you can write a loop in any of these three forms. For example, a while loop in (a) in the following figure can always be converted into the following for loop in (b):

```
while (loop-continuation-condition) {
  // Loop body
}
```

(a)

Equivalent

```
for ( ; loop-continuation-condition; )
  // Loop body
}
```

(b)

A for loop in (a) in the following figure can generally be converted into the following while loop in (b) except in certain special cases (see Review Question 3.19 for one of them):

```
for (initial-action;
     loop-continuation-condition;
     action-after-each-iteration) {
  // Loop body;
}
```

(a)

Equivalent

```
initial-action;
while (loop-continuation-condition) {
  // Loop body;
  action-after-each-iteration;
}
```

(b)

RECOMMENDATIONS

Use the one that is most intuitive and comfortable for you. In general, a for loop may be used if the number of repetitions is known, as, for example, when you need to print a message 100 times. A while loop may be used if the number of repetitions is not known, as in the case of reading the numbers until the input is 0. A do-while loop can be used to replace a while loop if the loop body has to be executed before testing the continuation condition.

NESTED LOOPS

Problem: Write a program that uses nested for loops to print a multiplication table.

MultiplicationTable

Run

MINIMIZING NUMERICAL ERRORS

Numeric errors involving floating-point numbers are inevitable. This section discusses how to minimize such errors through an example.

Here is an example that sums a series that starts with 0.01 and ends with 1.0. The numbers in the series will increment by 0.01, as follows: 0.01 + 0.02 + 0.03 and so on.

TestSum

Run

PROBLEM: FINDING THE GREATEST COMMON DIVISOR

Problem: Write a program that prompts the user to enter two positive integers and finds their greatest common divisor.

Solution: Suppose you enter two integers 4 and 2, their greatest common divisor is 2. Suppose you enter two integers 16 and 24, their greatest common divisor is 8. So, how do you find the greatest common divisor? Let the two input integers be n1 and n2. You know number 1 is a common divisor, but it may not be the greatest commons divisor. So you can check whether k (for k = 2, 3, 4, and so on) is a common divisor for n1 and n2, until k is greater than n1 or n2.

GreatestCommonDivisor

Run

PROBLEM: PREDICTING THE FUTURE TUITION

Problem: Suppose that the tuition for a university is $10,000 this year and tuition increases 7% every year. In how many years will the tuition be doubled?

FutureTuition

Run

PROBLEM: PREDICATING THE FUTURE TUITION

```
double tuition = 10000;   int year = 0  // Year 0
tuition = tuition * 1.07; year++;       // Year 1
tuition = tuition * 1.07; year++;       // Year 2
tuition = tuition * 1.07; year++;       // Year 3
...
```

CASE STUDY: *CONVERTING DECIMALS TO HEXADECIMALS*

Hexadecimals are often used in computer systems programming (see Appendix F for an introduction to number systems). How do you convert a decimal number to a hexadecimal number? To convert a decimal number d to a hexadecimal number is to find the hexadecimal digits h_n, h_{n-1}, h_{n-2}, ... , h_2, h_1, and h_0 such that

$$d = h_n \times 16^n + h_{n-1} \times 16^{n-1} + h_{n-2} \times 16^{n-2} + ... + h_2 \times 16^2 + h_1 \times 16^1 + h_0 \times 16^0$$

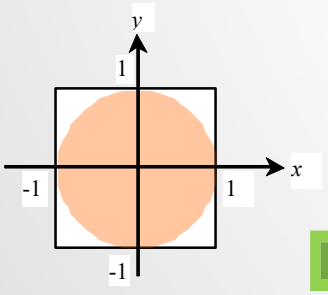
These hexadecimal digits can be found by successively dividing d by 16 until the quotient is 0. The remainders are h_0, h_1, h_2, ... , h_{n-2}, h_{n-1}, and h_n.

Dec2Hex Run

Companion Website

PROBLEM: *MONTE CARLO SIMULATION*

The Monte Carlo simulation refers to a technique that uses random numbers and probability to solve problems. This method has a wide range of applications in computational mathematics, physics, chemistry, and finance. This section gives an example of using the Monto Carlo simulation for estimating π.



circleArea / squareArea = π / 4.

π can be approximated as 4 * numberOfHits / numberOfTrials

MonteCarloSimulation Run

CONTROLLING LOOPS

- continue
  - Returns to loop condition without executing remainder of loop statements
- break
  - Exits loop immediately
- Only related to current inner loop
  - i.e. break will only exit current inner loop it is a part of, not an outer loop (if there exists an outer loop)

BREAK

```
public class TestBreak {
  public static void main(String[] args) {
    int sum = 0;
    int number = 0;

    while (number < 20) {
      number++;
      sum += number;
      if (sum >= 100)
        break;
    }

    System.out.println("The number is " + number);
    System.out.println("The sum is " + sum);
  }
}
```

CONTINUE

```
public class TestContinue {
  public static void main(String[] args) {
    int sum = 0;
    int number = 0;

    while (number < 20) {
      number++;
      if (number == 10 || number == 11)
        continue;
      sum += number;
    }

    System.out.println("The sum is " + sum);
  }
}
```

USING BREAK AND CONTINUE

Examples for using the `break` and `continue` keywords:

- TestBreak.java

- TestContinue.java

GUESSING NUMBER PROBLEM REVISITED

Here is a program for guessing a number. You can rewrite it using a break statement.

GuessNumberUsingBreak

Run

PROBLEM: DISPLAYING PRIME NUMBERS

Problem: Write a program that displays the first 50 prime numbers in five lines, each of which contains 10 numbers. An integer greater than 1 is *prime* if its only positive divisor is 1 or itself. For example, 2, 3, 5, and 7 are prime numbers, but 4, 6, 8, and 9 are not.

Solution: The problem can be broken into the following tasks:

- For number = 2, 3, 4, 5, 6, ..., test whether the number is prime.
- Determine whether a given number is prime.
- Count the prime numbers.
- Print each prime number, and print 10 numbers per line.

PrimeNumber

Run

CSE 101 - COMPUTER PROGRAMMING I METHODS

Joseph LEDET
Department of Computer Engineering
Akdeniz University
josephledet@akdeniz.edu.tr

OPENING PROBLEM

Find the sum of integers from 1 to 10, from 20 to 30, and from 35 to 45, respectively.

PROBLEM

```
int sum = 0;
for (int i = 1; i <= 10; i++)
  sum += i;
System.out.println("Sum from 1 to 10 is " + sum);

sum = 0;
for (int i = 20; i <= 30; i++)
  sum += i;
System.out.println("Sum from 20 to 30 is " + sum);

sum = 0;
for (int i = 35; i <= 45; i++)
  sum += i;
System.out.println("Sum from 35 to 45 is " + sum);
```

PROBLEM

```
int sum = 0;
for (int i = 1; i <= 10; i++)
  sum += i;
System.out.println("Sum from 1 to 10 is " + sum);

sum = 0;
for (int i = 20; i <= 30; i++)
  sum += i;
System.out.println("Sum from 20 to 30 is " + sum);

sum = 0;
for (int i = 35; i <= 45; i++)
  sum += i;
System.out.println("Sum from 35 to 45 is " + sum);
```

SOLUTION

```
public static int sum(int i1, int i2) {
  int sum = 0;
  for (int i = i1; i <= i2; i++)
    sum += i;
  return sum;
}

public static void main(String[] args) {
  System.out.println("Sum from 1 to 10 is " + sum(1, 10));
  System.out.println("Sum from 20 to 30 is " + sum(20, 30));
  System.out.println("Sum from 35 to 45 is " + sum(35, 45));
}
```


OBJECTIVES

- To define methods with formal parameters (§6.2).
- To invoke methods with actual parameters (i.e., arguments) (§6.2).
- To define methods with a return value (§6.3).
- To define methods without a return value (§6.4).
- To pass arguments by value (§6.5).
- To develop reusable code that is modular, easy to read, easy to debug, and easy to maintain (§6.6).
- To write a method that converts hexadecimals to decimals (§6.7).
- To use method overloading and understand ambiguous overloading (§6.8).
- To determine the scope of variables (§6.9).
- To apply the concept of method abstraction in software development (§6.10).
- To design and implement methods using stepwise refinement (§6.10).

DEFINING METHODS

A method is a collection of statements that are grouped together to perform an operation.

Define a method

```
public static int max(int num1, int num2) {
   int result;

   if (num1 > num2)
     result = num1;
   else
     result = num2;

   return result;
}
```

Invoke a method

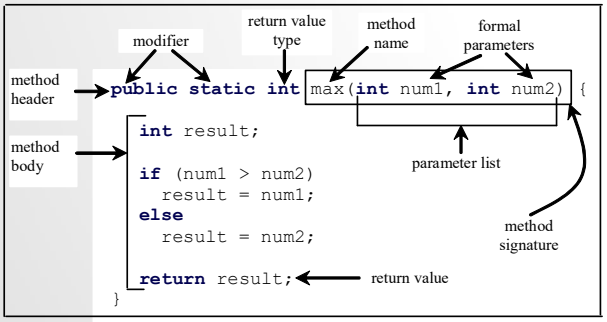
```
int z = max(x, y);
```

actual parameters (arguments)

DEFINING METHODS

A method is a collection of statements that are grouped together to perform an operation.

Define a method



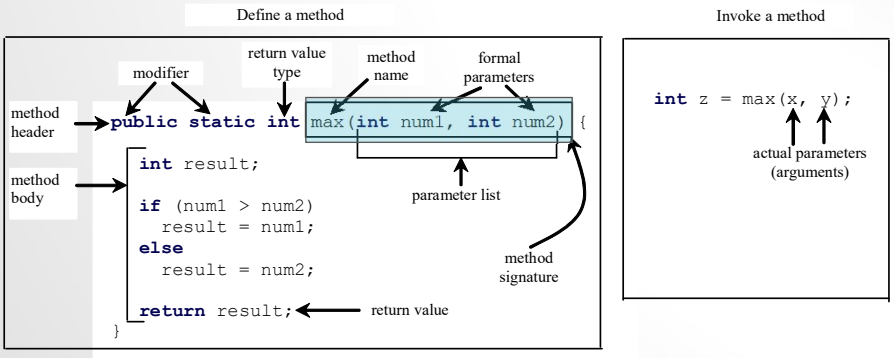
Invoke a method

```
int z = max(x, y);
```

actual parameters (arguments)

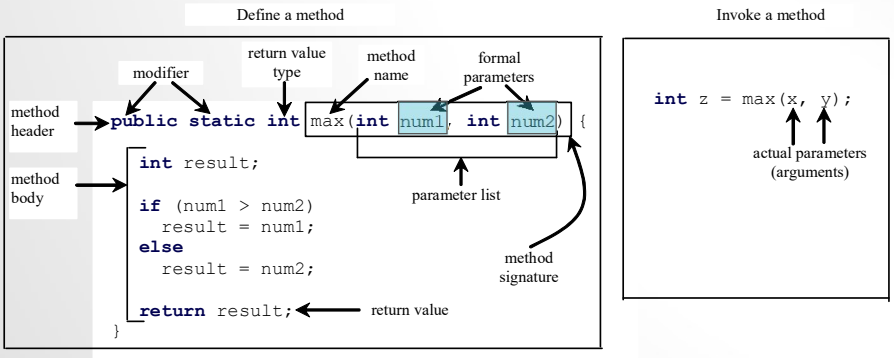
METHOD SIGNATURE

Method signature is the combination of the method name and the parameter list.



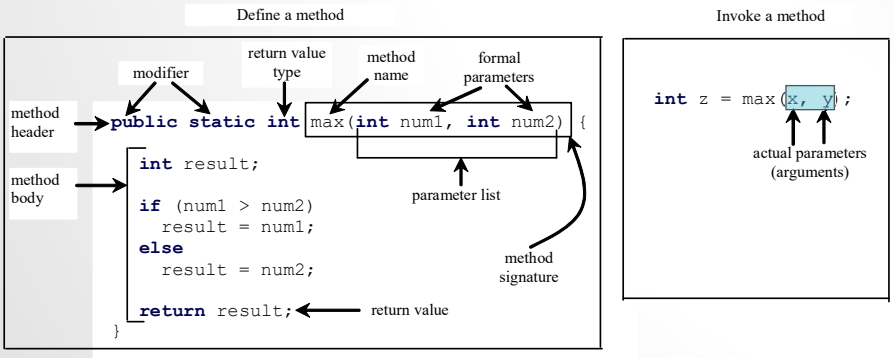
FORMAL PARAMETERS

The variables defined in the method header are known as *formal parameters*.



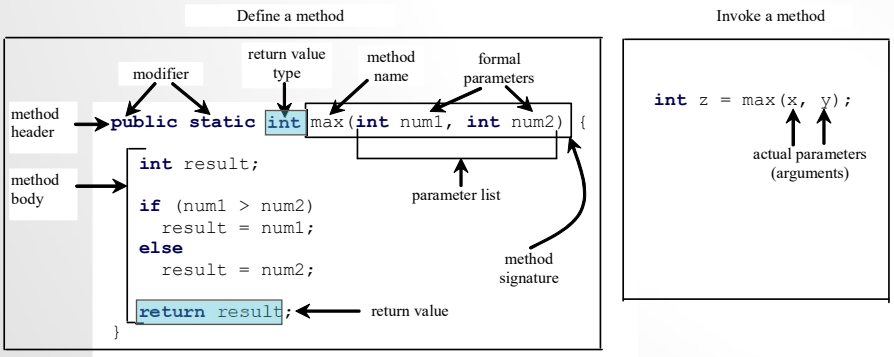
ACTUAL PARAMETERS

When a method is invoked, you pass a value to the parameter. This value is referred to as *actual parameter or argument*.



RETURN VALUE TYPE

A method may return a value. The returnValueType is the data type of the value the method returns. If the method does not return a value, the returnValueType is the keyword void. For example, the returnValueType in the main method is void.



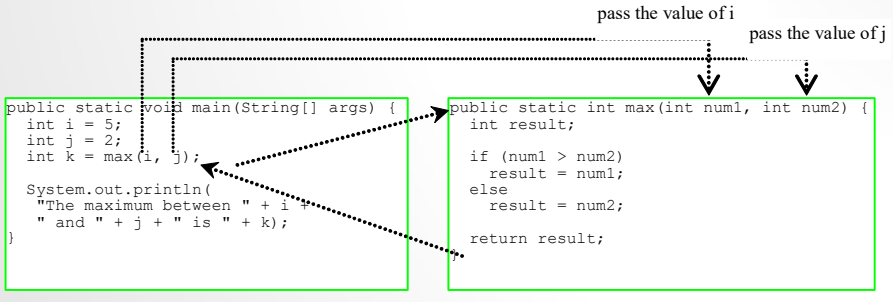
CALLING METHODS

Testing the `max` method

This program demonstrates calling a method max to return the largest of the `int` values

animation

CALLING METHODS, CONT.



animation

TRACE METHOD INVOCATION

i is now 5

```
public static void main(String[] args) {
  int i = 5;
  int j = 2;
  int k = max(i, j);

  System.out.println(
   "The maximum between " + i +
   " and " + j + " is " + k);
}
```

```
public static int max(int num1, int num2) {
  int result;

  if (num1 > num2)
    result = num1;
  else
    result = num2;

  return result;
}
```


animation

TRACE METHOD INVOCATION

j is now 2

```
public static void main(String[] args) {
  int i = 5;
  int j = 2;
  int k = max(i, j);

  System.out.println(
   "The maximum between " + i +
   " and " + j + " is " + k);
}
```

```
public static int max(int num1, int num2) {
  int result;

  if (num1 > num2)
    result = num1;
  else
    result = num2;

  return result;
}
```


animation

TRACE METHOD INVOCATION

invoke max(i, j)

```
public static void main(Strin   args) {
  int i = 5;
  int j = 2;
  int k = max(i, j);

  System.out.println(
   "The maximum between " + i +
   " and " + j + " is " + k);
}
```

```
public static int max(int num1, int num2) {
  int result;

  if (num1 > num2)
    result = num1;
  else
    result = num2;

  return result;
}
```

animation

TRACE METHOD INVOCATION

invoke max(i, j)
Pass the value of i to num1
Pass the value of j to num2

```
public static void main(String[] args) {
  int i = 5;
  int j = 2;
  int k = max(i, j);

  System.out.println(
   "The maximum between " + i +
   " and " + j + " is " + k);
}
```

```
public static int max(int num1, int num2) {
  int result;

  if (num1 > num2)
    result = num1;
  else
    result = num2;

  return result;
}
```

animation

TRACE METHOD INVOCATION

declare variable result

```
public static void main(String[] args) {
  int i = 5;
  int j = 2;
  int k = max(i, j);

  System.out.println(
   "The maximum between " + i +
   " and " + j + " is " + k);
}
```

```
public static int max(int num1, int num2) {
  int result;

  if (num1 > num2)
    result = num1;
  else
    result = num2;

  return result;
}
```


animation

TRACE METHOD INVOCATION

(num1 > num2) is true since num1 is 5 and num2 is 2

```
public static void main(String[] args) {
  int i = 5;
  int j = 2;
  int k = max(i, j);

  System.out.println(
   "The maximum between " + i +
   " and " + j + " is " + k);
}
```

```
public static int max(int num1, int num2) {
  int result;

  if (num1 > num2)
    result = num1;
  else
    result = num2;

  return result;
}
```


animation

TRACE METHOD INVOCATION

result is now 5

```
public static void main(String[] args) {
  int i = 5;
  int j = 2;
  int k = max(i, j);

  System.out.println(
   "The maximum between " + i +
   " and " + j + " is " + k);
}
```

```
public static int max(int num1, int num2) {
  int result;

  if (num1 > num2)
    result = num1;
  else
    result = num2;

  return result;
}
```


animation

TRACE METHOD INVOCATION

return result, which is 5

```
public static void main(String[] args) {
  int i = 5;
  int j = 2;
  int k = max(i, j);

  System.out.println(
   "The maximum between " + i +
   " and " + j + " is " + k);
}
```

```
pu    tatic int max(int num1, int num2) {
      esult;

    num1 > num2)
    esult = num1;
  se
    result = num2;

  return result;
}
```


animation

TRACE METHOD INVOCATION

return max(i, j) and assign the return value to k

```
public static void main(String[] args) {
  int i = 5;
  int j = 2;
  int k = max(i, j);

  System.out.println(
    "The maximum between " + i +
    " and " + j + " is " + k);
}
```

```
public static int max(int num1, int num2) {
  int result;

  if (num1 > num2)
    result = num1;
  else
    result = num2;

  return result;
}
```

animation

TRACE METHOD INVOCATION

Execute the print statement

```
public static void main(String[] args) {
  int i = 5;
  int j = 2;
  int k = max(i, j);

  System.out.println(
    "The maximum between " + i +
    " and " + j + " is " + k);
}
```

```
public static int max(int num1, int num2) {
  int result;

  if (num1 > num2)
    result = num1;
  else
    result = num2;

  return result;
}
```

CAUTION

A return statement is required for a value-returning method. The method shown below in (a) is logically correct, but it has a compilation error because the Java compiler thinks it possible that this method does not return any value.

```
public static int sign(int n) {
  if (n > 0)
    return 1;
  else if (n == 0)
    return 0;
  else if (n < 0)
    return -1;
}
```

(a)

Should be

```
public static int sign(int n) {
  if (n > 0)
    return 1;
  else if (n == 0)
    return 0;
  else
    return -1;
}
```

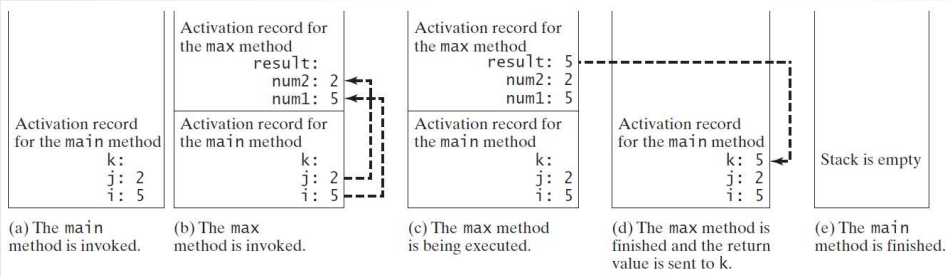
(b)

To fix this problem, delete *if (n < 0)* in (a), so that the compiler will see a return statement to be reached regardless of how the if statement is evaluated.

REUSE METHODS FROM OTHER CLASSES

NOTE: One of the benefits of methods is for reuse. The max method can be invoked from any class besides TestMax. If you create a new class Test, you can invoke the max method using ClassName.methodName (e.g., TestMax.max).

CALL STACKS



(a) The `main` method is invoked. (b) The `max` method is invoked. (c) The `max` method is being executed. (d) The `max` method is finished and the return value is sent to k. (e) The `main` method is finished.

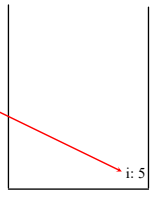
animation

TRACE CALL STACK

i is declared and initialized

```
public static void main(String[] args) {
  int i = 5;
  int j = 2;
  int k = max(i, j);

  System.out.println(
    "The maximum between " + i +
    " and " + j + " is " + k);
}
```

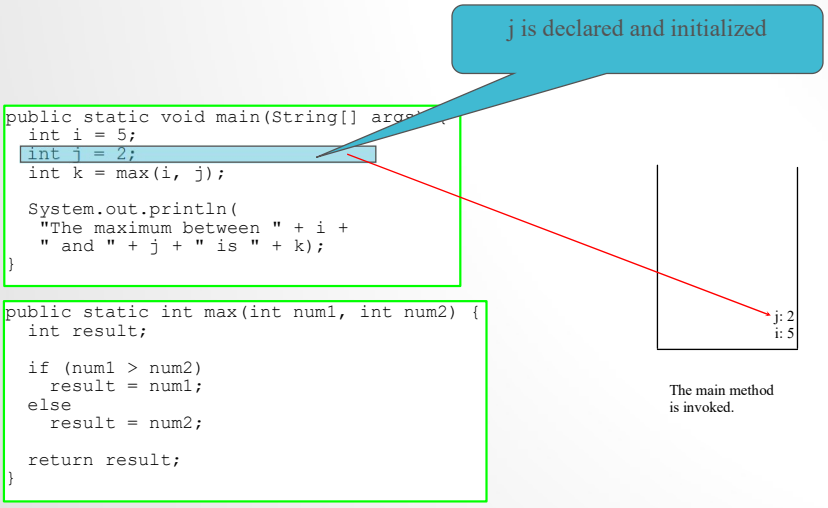
```
public static int max(int num1, int num2) {
  int result;

  if (num1 > num2)
    result = num1;
  else
    result = num2;

  return result;
}
```



The main method is invoked.

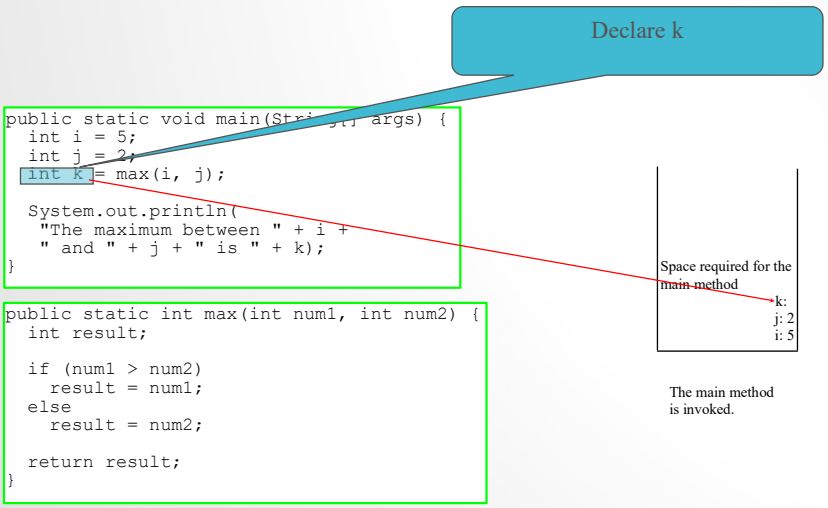
animation

TRACE CALL STACK



animation

TRACE CALL STACK



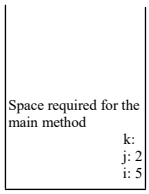
animation

TRACE CALL STACK

Invoke max(i, j)

```
public static void main(String[] args) {
  int i = 5;
  int j = 2;
  int k = max(i, j);

  System.out.println(
   "The maximum between " + i +
   " and " + j + " is " + k);
}
```

```
public static int max(int num1, int num2) {
  int result;

  if (num1 > num2)
    result = num1;
  else
    result = num2;

  return result;
}
```



The main method is invoked.

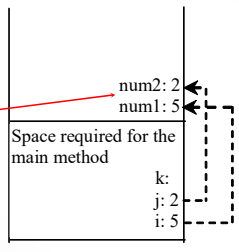
animation

TRACE CALL STACK

pass the values of i and j to num1 and num2

```
public static void main(String[] args) {
  int i = 5;
  int j = 2;
  int k = max(i, j);

  System.out.println(
   "The maximum between " + i +
   " and " + j + " is " + k);
}
```

```
public static int max(int num1, int num2) {
  int result;

  if (num1 > num2)
    result = num1;
  else
    result = num2;

  return result;
}
```



The max method is invoked.

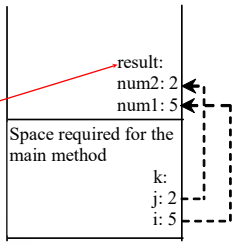
animation

TRACE CALL STACK

Declare result

```
public static void main(String[] args) {
  int i = 5;
  int j = 2;
  int k = max(i, j);

  System.out.println(
   "The maximum between " + i +
   " and " + j + " is " + k);
}
```

```
public static int max(int num1, int num2) {
  int result;

  if (num1 > num2)
    result = num1;
  else
    result = num2;

  return result;
}
```



The max method is invoked.

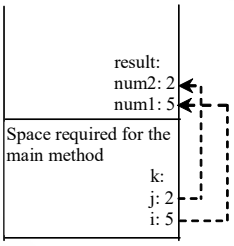
animation

TRACE CALL STACK

(num1 > num2) is true

```
public static void main(String[] args) {
  int i = 5;
  int j = 2;
  int k = max(i, j);

  System.out.println(
   "The maximum between " + i +
   " and " + j + " is " + k);
}
```

```
public static int max(int num1, int num2) {
  int result;

  if (num1 > num2)
    result = num1;
  else
    result = num2;

  return result;
}
```



The max method is invoked.

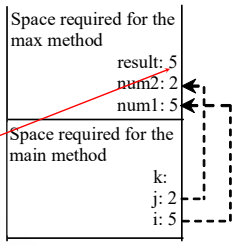
animation

TRACE CALL STACK

Assign num1 to result

```
public static void main(String[] args) {
  int i = 5;
  int j = 2;
  int k = max(i, j);

  System.out.println(
   "The maximum between " + i +
   " and " + j + " is " + k);
}
```

```
public static int max(int num1, int num2)
  int result;

  if (num1 > num2)
    result = num1;
  else
    result = num2;

  return result;
}
```



The max method is invoked.

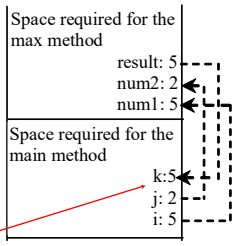
animation

TRACE CALL STACK

Return result and assign it to k

```
public static void main(String[] args) {
  int i = 5;
  int j = 2;
  int k = max(i, j);

  System.out.println(
   "The maximum between " + i +
   " and " + j + " is " + k);
}
```

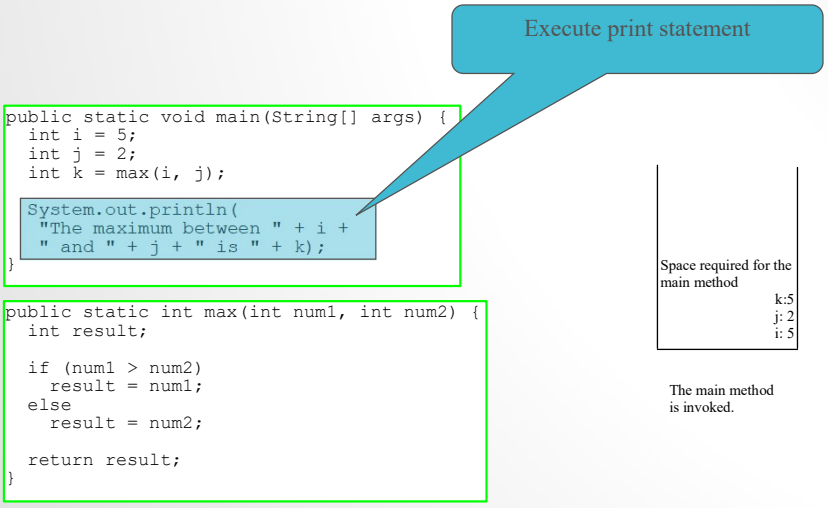
```
public static int max(int num1, int num2
  int result;

  if (num1 > num2)
    result = num1;
  else
    result = num2;

  return result;
}
```



The max method is invoked.

animation

TRACE CALL STACK



VOID METHOD EXAMPLE

This type of method does not return a value. The method performs some actions.

TestVoidMethod Run

TestReturnGradeMethod Run

PASSING PARAMETERS

```
public static void nPrintln(String message, int n) {
  for (int i = 0; i < n; i++)
    System.out.println(message);
}
```

Suppose you invoke the method using
nPrintln("Welcome to Java", 5);
What is the output?

Suppose you invoke the method using
nPrintln("Computer Science", 15);
What is the output?

Can you invoke the method using
nPrintln(15, "Computer Science");

PASS BY VALUE

This program demonstrates passing values to the methods.

Increment

Run

PASS BY VALUE

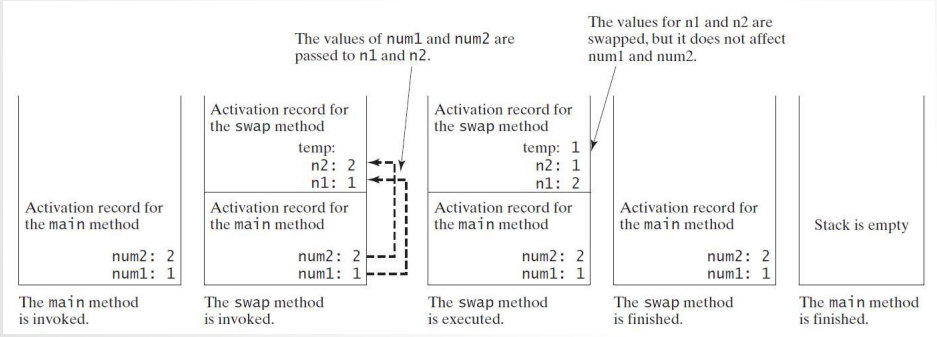
Testing Pass by value

This program demonstrates passing values to the methods.

TestPassByValue

Run

PASS BY VALUE, CONT.



MODULARIZING CODE

Methods can be used to reduce redundant coding and enable code reuse. Methods can also be used to modularize code and improve the quality of the program.

GreatestCommonDivisorMethod

Run

PrimeNumberMethod

Run

CASE STUDY: CONVERTING HEXADECIMALS TO DECIMALS

Write a method that converts a hexadecimal number into a decimal number.

ABCD =>

A*16^3 + B*16^2 + C*16^1+ D*16^0

= ((A*16 + B)*16 + C)*16+D

= ((10*16 + 11)*16 + 12)*16+13 = ?

Hex2Dec

Run

OVERLOADING METHODS

Overloading the `max` Method

```
public static double max(double num1, double
 num2) {
  if (num1 > num2)
    return num1;
  else
    return num2;
}
```

TestMethodOverloading

Run

AMBIGUOUS INVOCATION

Sometimes there may be two or more possible matches for an invocation of a method, but the compiler cannot determine the most specific match. This is referred to as *ambiguous invocation*. Ambiguous invocation is a compile error.

AMBIGUOUS INVOCATION

```
public class AmbiguousOverloading {
  public static void main(String[] args) {
    System.out.println(max(1, 2));
  }

  public static double max(int num1, double num2) {
    if (num1 > num2)
      return num1;
    else
      return num2;
  }

  public static double max(double num1, int num2) {
    if (num1 > num2)
      return num1;
    else
      return num2;
  }
}
```


SCOPE OF LOCAL VARIABLES

A local variable: a variable defined inside a method.

Scope: the part of the program where the variable can be referenced.

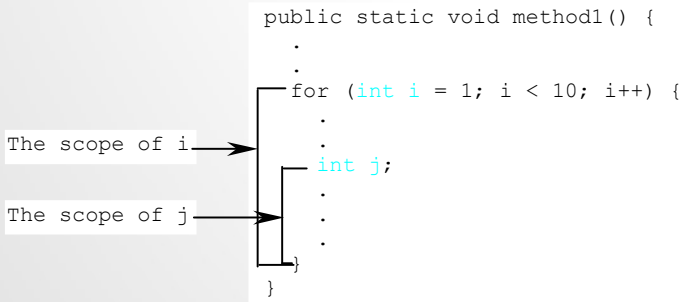
The scope of a local variable starts from its declaration and continues to the end of the block that contains the variable. A local variable must be declared before it can be used.

SCOPE OF LOCAL VARIABLES, CONT.

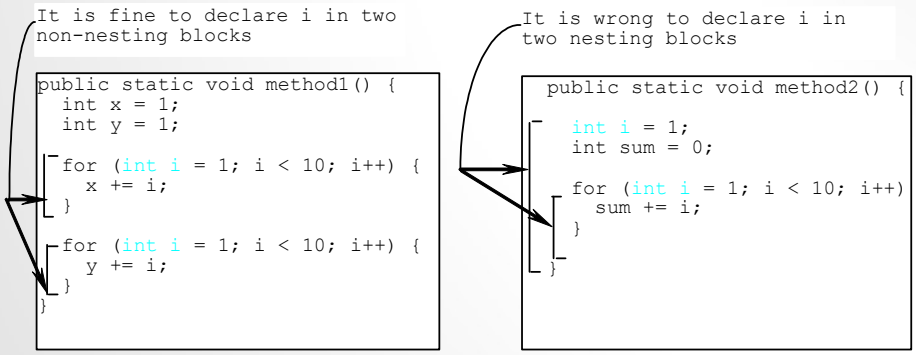
You can declare a local variable with the same name multiple times in different non-nesting blocks in a method, but you cannot declare a local variable twice in nested blocks.

SCOPE OF LOCAL VARIABLES, CONT.

A variable declared in the initial action part of a for loop header has its scope in the entire loop. But a variable declared inside a for loop body has its scope limited in the loop body from its declaration and to the end of the block that contains the variable.



SCOPE OF LOCAL VARIABLES, CONT.



SCOPE OF LOCAL VARIABLES, CONT.

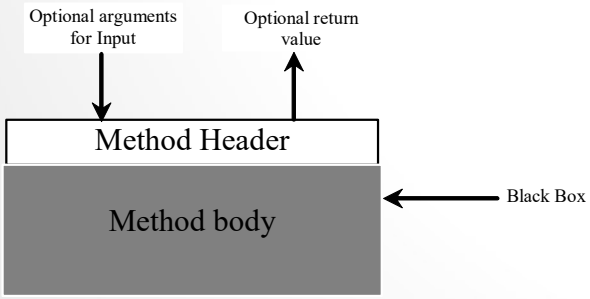
```
// Fine with no errors
public static void correctMethod() {
  int x = 1;
  int y = 1;
  // i is declared
  for (int i = 1; i < 10; i++) {
    x += i;
  }
  // i is declared again
  for (int i = 1; i < 10; i++) {
    y += i;
  }
}
```

SCOPE OF LOCAL VARIABLES, CONT.

```
// With errors
public static void incorrectMethod() {
  int x = 1;
  int y = 1;
  for (int i = 1; i < 10; i++) {
    int x = 0;
    x += i;
  }
}
```

METHOD ABSTRACTION

You can think of the method body as a black box that contains the detailed implementation for the method.



BENEFITS OF METHODS

- Write a method once and reuse it anywhere.
- Information hiding. Hide the implementation from the user.
- Reduce complexity.

CASE STUDY: GENERATING RANDOM CHARACTERS

Computer programs process numerical data and characters. You have seen many examples that involve numerical data. It is also important to understand characters and how to process them.

As introduced in Section 2.9, each character has a unique Unicode between 0 and FFFF in hexadecimal (65535 in decimal). To generate a random character is to generate a random integer between 0 and 65535 using the following expression: (note that since 0 <= Math.random() < 1.0, you have to add 1 to 65535.)

```
(int)(Math.random() * (65535 + 1))
```

CASE STUDY: GENERATING RANDOM CHARACTERS, CONT.

Now let us consider how to generate a random lowercase letter. The Unicode for lowercase letters are consecutive integers starting from the Unicode for 'a', then for 'b', 'c', ..., and 'z'. The Unicode for 'a' is

```
(int)'a'
```

So, a random integer between (int)'a' and (int)'z' is

```
(int)((int)'a' + Math.random() * ((int)'z' - (int)'a' + 1)
```

CASE STUDY: GENERATING RANDOM CHARACTERS, CONT.

As discussed in Chapter 2., all numeric operators can be applied to the char operands. The char operand is cast into a number if the other operand is a number or a character. So, the preceding expression can be simplified as follows:

```
'a' + Math.random() * ('z' - 'a' + 1)
```

So a random lowercase letter is

```
(char)('a' + Math.random() * ('z' - 'a' + 1))
```

CASE STUDY: GENERATING RANDOM CHARACTERS, CONT.

To generalize the foregoing discussion, a random character between any two characters ch1 and ch2 with ch1 < ch2 can be generated as follows:

(char)(ch1 + Math.random() * (ch2 – ch1 + 1))

THE RANDOMCHARACTER CLASS

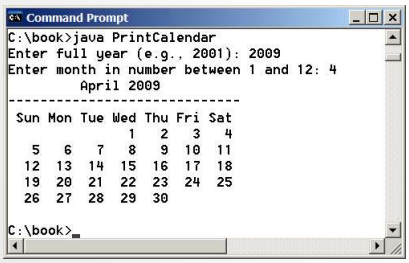
```
// RandomCharacter.java: Generate random characters
public class RandomCharacter {
  /** Generate a random character between ch1 and ch2 */
  public static char getRandomCharacter(char ch1, char ch2) {
    return (char)(ch1 + Math.random() * (ch2 - ch1 + 1));
  }

  /** Generate a random lowercase letter */
  public static char getRandomLowerCaseLetter() {
    return getRandomCharacter('a', 'z');
  }

  /** Generate a random uppercase letter */
  public static char getRandomUpperCaseLetter() {
    return getRandomCharacter('A', 'Z');
  }

  /** Generate a random digit character */
  public static char getRandomDigitCharacter() {
    return getRandomCharacter('0', '9');
  }

  /** Generate a random character */
  public static char getRandomCharacter() {
    return getRandomCharacter('\u0000', '\uFFFF');
  }
}
```

STEPWISE REFINEMENT (OPTIONAL)

The concept of method abstraction can be applied to the process of developing programs. When writing a large program, you can use the "divide and conquer" strategy, also known as *stepwise refinement*, to decompose it into subproblems. The subproblems can be further decomposed into smaller, more manageable problems.

PRINTCALENDER CASE STUDY

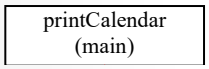
Let us use the PrintCalendar example to demonstrate the stepwise refinement approach.



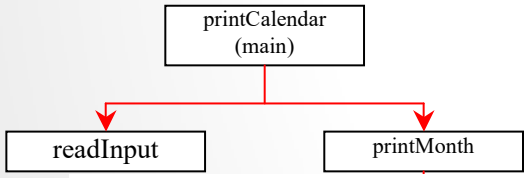
PrintCalendar

Run

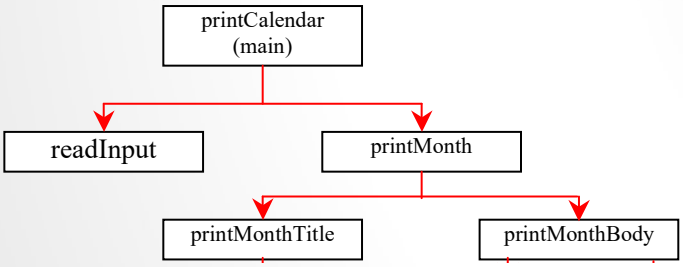
DESIGN DIAGRAM



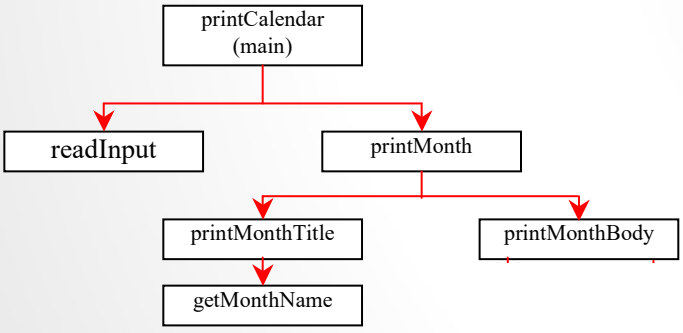
DESIGN DIAGRAM



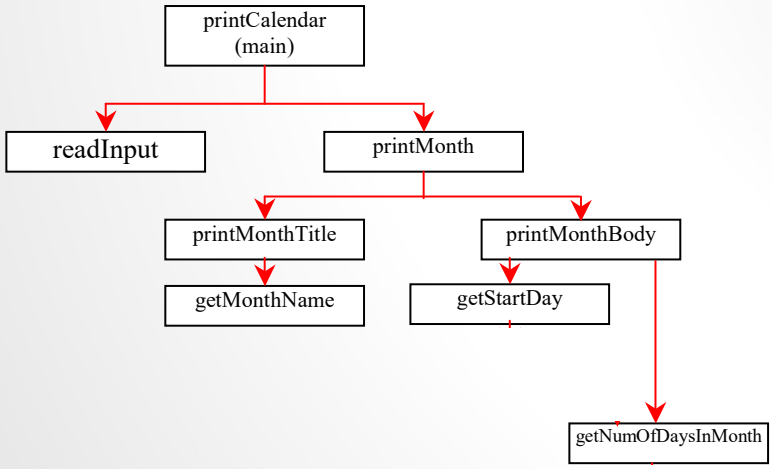
DESIGN DIAGRAM



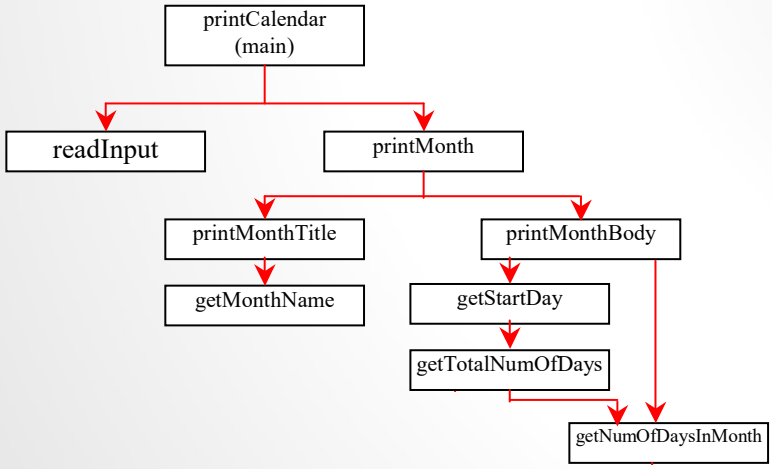
DESIGN DIAGRAM



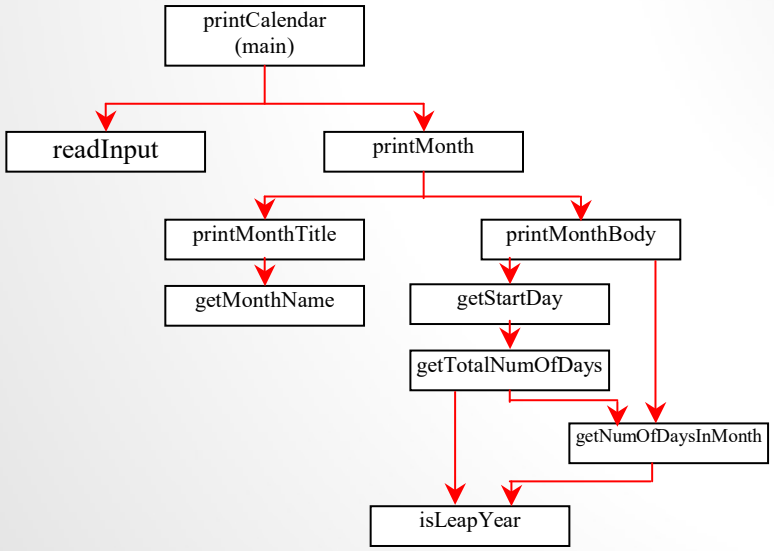
DESIGN DIAGRAM



DESIGN DIAGRAM



DESIGN DIAGRAM



IMPLEMENTATION: TOP-DOWN

Top-down approach is to implement one method in the structure chart at a time from the top to the bottom. Stubs can be used for the methods waiting to be implemented. A stub is a simple but incomplete version of a method. The use of stubs enables you to test invoking the method from a caller. Implement the main method first and then use a stub for the printMonth method. For example, let printMonth display the year and the month in the stub. Thus, your program may begin like this:

A Skeleton for printCalendar

IMPLEMENTATION: BOTTOM-UP

Bottom-up approach is to implement one method in the structure chart at a time from the bottom to the top. For each method implemented, write a test program to test it. Both top-down and bottom-up methods are fine. Both approaches implement the methods incrementally and help to isolate programming errors and makes debugging easy. Sometimes, they can be used together.

BENEFITS OF STEPWISE REFINEMENT

Simpler Program

Reusing Methods

Easier Developing, Debugging, and Testing

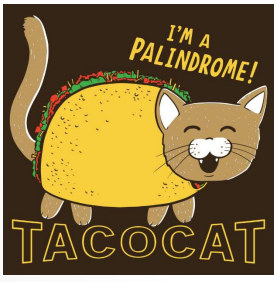
Better Facilitating Teamwork

SOME EXAMPLES

- Palindrome
  - String that is the same forward and backward
- Number Rectangle
  - Given two integers (`row` and `col`), print `row` rows of the numbers 1 through `col`
- Substring to a character
  - Given a String and a character, get the String up to the first time that character appears
- Count characters
  - Get the number of times that a character appears in a String
- Menu
  - Display a set of menu options

PALINDROME

- A text that is the same forward and backward
- racecar 
- mom 
- kayak 
- noon 
- tacocat 



NUMBER RECTANGLE

- N rows of the numbers 1 to M
- 5 and 5

1	2	3	4	5
1	2	3	4	5
1	2	3	4	5
1	2	3	4	5
1	2	3	4	5

2 and 3

1	2	3
1	2	3

STRING METHODS

- Substring to a character
  - Get the characters in a string up to the first time a certain character appears
  - "CSE 101 is awesome!" and 'm' gives "CSE 101 is aweso"
- Count the number of times a character appears in a string
  - "bir berber bir berbere" and 'b' gives 6

MENU

- A method that only displays a set of menu options
- Consider how to return a value from the menu