

CSE 201 Course Packet

CSE 201: Elementary Computer Programming Home Page:

<http://www.cse.ohio-state.edu/cse201>

Table of Contents

Note: page numbers below refer to “CSE 201 Course Packet” page numbers at the bottom of each page. It is recommended that you attach physical tabs to the pages mentioned for easy navigation.

Tab 1: Policies	3
Syllabus, course policies, grading criteria, and other useful information	
Tab 2: Slides	13
Copies of the Powerpoint slides used in class	
Tab 3: Code	89
Code examples used in class	

CSE 201: Elementary Computer Programming

Description

Introduction to computer programming and to problem solving techniques using computer programs; programming lab experience.

Level and Credits

- U 4 (three one-hour lectures, one one-hour lab)

Prerequisites

- None.

Quarters Offered

- Au, Wi, Sp, Su

General Information, Exclusions, etc.

- Java is taught.

Objectives

- Master using basic coding features provided by high-level imperative programming languages;
- Master writing computer programs to implement given simple algorithms;
- Be familiar with analyzing simple real-life problems and choosing appropriate algorithms for their solution;
- Be familiar with using basic data structures such as arrays in simple programs;
- Be familiar with using methods and classes to help produce well-structured programs;
- Be familiar with reading and programming for API's;
- Be familiar with designing simple text-oriented user interfaces;
- Be familiar with working in a window-based computing environment;
- Be exposed to the services provided by an operating system;
- Be exposed to the virtual machine model of modern computer systems;
- Be exposed to data abstraction concepts and other more advanced programming ideas.

Course Web Page

- Up-to-date information for the course is accessible online at <http://www.cse.ohio-state.edu/cse201>.

Texts

- Tony Gaddis, *Starting Out with Java: From Control Structures through Objects* (3rd ed. or later), Pearson Education/Addison Wesley, 2008, ISBN 0-321-47927-0.
- *CSE 201 Course Notes*, OSU UniPrint, updated quarterly.

Topics (Approximate)

Number of Weeks	Topics
1	Course introduction and basic concepts
1	Primitive types and expressions; String; basic I/O
3	Flow of control and Boolean expressions
1.5	Defining methods
1.5	Arrays
1	Basic exception handling and standard Java I/O

Lab Assignments

1. Environment walkthrough
2. Primitive types, assignment, arithmetic expressions, simple I/O
3. Control structures I
4. Control structures II
5. Methods I
6. Methods II/Arrays of primitive values
7. Methods III/Arrays of reference objects
8. Standard I/O

Grading Plan

Midterm Exam	20%
Final Exam	30%
Homework Assignments	10%
Lab Assignments	35%
Class Participation	5%

Important Note: A passing grade on the final exam is *required* in order to receive a passing grade for the course.

CSE 201 Course Policies

Electronic Mail

You will be expected to be able to use *electronic mail* to communicate personally with your instructor or grader or anyone else. ***Please get in the habit of checking your e-mail once every day or two.***

When sending e-mail to your instructor or grader, please be aware that "spam" filters may intercept e-mail sent from an external account (e.g., sally@aol.com or bill@hotmail.com) and either delay it or prevent it from reaching your intended recipient. We therefore strongly recommend sending e-mail from your OSU account.

Web Pages

You will be expected to be able to use a *web browser* to access the Internet. Essentially all course-related information will be available on-line through the appropriate course home page:

- <http://www.cse.ohio-state.edu/cse201>

For example, the course syllabus, all assignments, etc., are available this way. Some materials and assignments will appear incrementally during the course of the quarter.

Tutor Room

A tutor room will be available to provide additional help for students currently enrolled in CSE 201. The room is located in DL 299 and the schedule may be accessed at <http://www.cse.ohio-state.edu/cse201/tutor.html>. The room will be staffed by the graduate teaching associates currently teaching CSE 201, in lieu of individual office hours. You may visit the room at any time it is open and you need assistance understanding the material or completing the labs.

If you need additional help understanding the material, don't hesitate to contact your instructor via email, phone, and/or in person, to schedule an appointment.

Computing

The CSE Department has two computer labs where you can access the Java compiler. They are located in Caldwell Lab 112 and Baker 310. The labs are only open at certain times and the schedule is available at <http://www.cse.ohio-state.edu/cs/labs.shtml>. Caldwell Lab 112 is generally staffed with a Java consultant.

During the hours the lab is open, a monitor or operator will be available to help with any machine problems. In addition, the lab is staffed with a consultant to help with problems in using the operating system, the editor, the debugger, and syntax questions. If the consultant suggests radical changes or

changes you feel are not appropriate, use common sense and check with the instructor before making changes. Note that the consultants are not supposed to write portions of your program for you.

You may also choose to work from home if you do not require any assistance from the consultants. However, you will have to install the appropriate software on your personal computer. The course instructors will not be able to assist you in setting up this sort of thing; which is not to say it is impossible, only that you shouldn't expect the instructors to help.

The CSE Computing Services Help Desk staff are responsible for all the computers and your CSE accounts. If you have a problem with your password or username, visit them on the 8th floor of Dreese (DL 895, 292-6542).

Exams

If you expect to be unavailable for an exam, please make alternate arrangements *in advance*. You will need a documented, valid excuse for missing an exam. If you have trouble writing in English (and this goes for native English speakers as well as foreign students), practice! Points will be deducted for incomprehensible answers -- probably more than for wrong ones.

Please note the following important statement in the course syllabus: "A passing grade on the final exam is *required* in order to receive a passing grade for the course."

Assignments

Homework and other written assignments should be done in a "professional" manner. This does not mean "expensive". It means that you should make your reports look nice in addition to having the right content. You are strongly encouraged to use a word processor to prepare them.

In addition, we will have specific design and coding standards, just like most professional programming organizations. Adherence to these standards will be worth a substantial fraction of each lab assignment.

Be sure to keep all graded material until you have received your final grade in the course.

Late-Work Policies

- Homework assignments are due at the *start of class* on the due date. Late homework will not be accepted. If you will not be able to attend class on the day a homework is due, make arrangements with your instructors to turn in your homework *before* the due date.
- Lab (programming) assignments are due at the deadline time on the due date. A penalty of 25% will be assessed for a lab submitted one day (or part thereof) late, 50% will be assessed for two days (or part thereof) late, after which the lab will not be accepted. The final lab may not be turned in late. Labs are due by 7:00 pm on the due date.

Cooperation, Collaboration, and Professional Ethics

The policy on collaboration with others is fairly liberal -- but please don't be tempted to test its limits. Certain things clearly will be permissible (e.g., discussing problems and solution approaches) and certain things clearly will not be permissible (e.g., passing off as your own the work of someone else). Some people think there is a fuzzy area in between. If you have doubts about the middle area, stay out of it; ask your instructor for assistance. Violations are surprisingly easy to detect and they must and will be dealt with according to OSU rules on academic misconduct.

- You may ask the consultants or others for assistance with the computers, Windows, Eclipse, or Java (including interpreting the meanings of error messages and general advice about what causes them, but not including actually fixing the errors).
- You may *not* write or otherwise record any part of your solution to an assignment while someone is helping you.
- You may *not* take a physical or electronic copy of any part of a solution to an assignment from anyone.
- You may *not* give a physical or electronic copy of any part of a solution to an assignment to anyone.

One possibly ambiguous area involves talking to others about homework assignments and about the design, logic, and implementation of a program. You are *encouraged* to talk with others (especially others in the class) about these things. But do not give anyone or take from anyone written or recorded material, and in all cases please ***write up your own solution without assistance***. If you feel the need to cheat on these rules or are not sure whether some activity would constitute cheating, please discuss your questions with your instructor first!

And just in case it is not clear from the statements above:

- You may *not* ask for any help on the internet to solve any assignments (homeworks and labs).
- You may *not* search on the internet for any solutions to any of the assignments.
- You may *not* use any partial or complete solution found on the internet to any of the assignments for any reason or purpose.

There is one other rule about professional ethics:

- You may *not* turn in an assignment solution from a previous quarter's offering of the course.

Please note that this last rule applies *even if* you have previously taken the course and you think it might save you some time to turn in an old solution. Lab assignments may change in subtle ways from one quarter to the next. Any homework or lab submission that gives evidence of having been prepared for a previous quarter's course offering will receive zero credit. Moreover, if there is reason to suspect you got the questionable solution from someone else who took the course in a previous quarter, it will be treated as academic misconduct just as if you had gotten it from someone else who is taking the course this quarter.

For other information about appropriate use of the laboratory computing facilities, please see the

official policies.

Accommodation for Disability

If you need an accommodation based on the impact of a disability, you should contact your instructor to arrange an appointment as soon as possible. At the appointment you and the instructor can discuss the course format, anticipate your needs and explore potential accommodations. We rely on the Office for Disability Services for assistance in verifying the need for accommodations and developing accommodation strategies. If you have not previously contacted the Office for Disability Services, we encourage you to do so.

CSE 201 Course Grading Criteria: Homework Assignments

Homeworks will be graded based on the following general criteria as well as specific problem-related criteria:

- The answers are easy to read and easy to understand, in the sense that the writing/printing is legible, English is well-written, diagrams are well-drawn, etc. (It generally helps to produce your solutions using a computer.)
- The answers are complete but concise and to-the-point.
- The answers are correct.
- Programs and program fragments meet the lab assignment grading criteria.
- The submission is generally of the *high quality* expected of a *professional*.

Each homework assignment will receive one of the following marks:

Mark	Meaning	Points (%)
5	the solution meets all criteria well	100
4	the solution meets most criteria, but there is some room for improvement	80
3	the solution is just satisfactory; it meets some criteria but there is significant room for improvement	60
2	the solution is barely acceptable; there are serious shortcomings in meeting most criteria; it needs a lot of improvement	40
0	the solution is not acceptable	0

CSE 201 Course Grading Criteria: Lab (Programming) Assignments

Lab assignments will be graded according to the following general criteria:

- The solution adequately addresses the problem at hand. Specifically:
 - The solution clearly represents a good-faith attempt to address the requirements for the assignment.
 - The program compiles, links, and executes.
 - The program runs correctly (or at least appears to be correct based on testing done by the grader).
- The solution constitutes a **high quality** product expected of a **professional**. Specifically:
 - The program is easy to read and to understand, i.e., it is well commented and adheres to the course standards for layout and format (see the Java Code Conventions for an example set of programming conventions). For example, method and object names are appropriate and thoughtfully chosen, and all potentially confusing/complex program code (including non-trivial if-statements, loops, method calls, etc.) is adequately documented.
 - The general design of the program is clear and reasonable. For example, the program makes good use of classes and methods; and the algorithms used are appropriate for the task, are clearly explained, and are implemented in a sensible, understandable way.
 - All procedure and function headers include comments explaining *what* the method is supposed to do (not *how* it does it) and the purpose of each formal parameter. Be as precise and careful as you can be.

For some assignments, additional criteria might be added to this list. In this case, the lab assignment or your instructor will point them out.

Each lab assignment will receive a grade according to the following table:

Quality of Submitted Solution	Points (%)
the solution meets all criteria well	100
the solution meets most criteria, but there is some room for improvement	80
the solution is just satisfactory; it meets some criteria but there is significant room for improvement	60
the solution is barely acceptable; there are serious shortcomings in meeting most criteria; it needs a lot of improvement	40
the solution is not acceptable	0

Any lab submitted containing syntax errors or producing garbage output (multiple copies of the same line, nonsense data, incorrect values or totals, etc.) will receive an **automatic** grade of zero. Also, it is

your responsibility to be sure that all files are submitted in a timely and thorough fashion. No credit will be given for labs written but not submitted.

The instructor and/or the course coordinator may at times question a student about his/her lab submissions. If the student is unable to explain satisfactorily his/her solution (what it does, how it does it, why it was designed that way, etc.), a (possibly substantial) grade penalty may be imposed.

Students often wonder why a seemingly small error can cause a relatively large grade deduction. The reason is that software that does not work properly even in the smallest detail -- not in these assignments, of course, but in the "real world" -- can be costly or even dangerous to its users; and software that is difficult to change can be costly to companies that develop it. Most employers of software professionals therefore have high standards for quality. We want you to get used to this: to understand that "almost right" isn't good enough for most employers, and that as a professional it shouldn't be good enough for you even if it were good enough for your employer. Therefore, an error that prevents your program from working (especially from compiling properly so it can run at all) can result in a significant grade deduction.

CSE 201

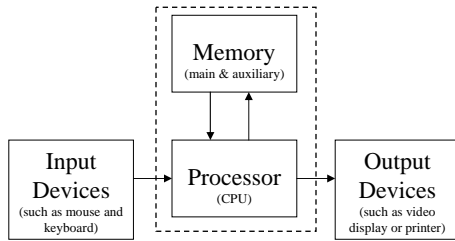
Elementary Computer Programming

Computer Basics

- Computer system: hardware + software
- Hardware: the physical components
- Software: the instructions that tell the hardware what to do

Common Hardware Components

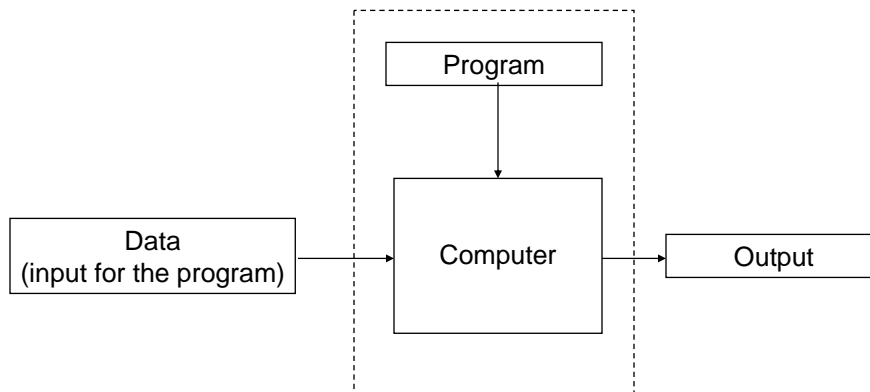
Standard Hardware Organization



- Processor (CPU)
 - Central Processing Unit
 - Interprets and executes the instructions
- Memory
 - main & auxiliary
 - holds data and instructions
- Input device(s)
 - mouse, keyboard, etc.
- Output device(s)
 - video display, printer, etc.
- CPU and memory are physically housed together

Running a Program

Program—a set of instructions for a computer to follow



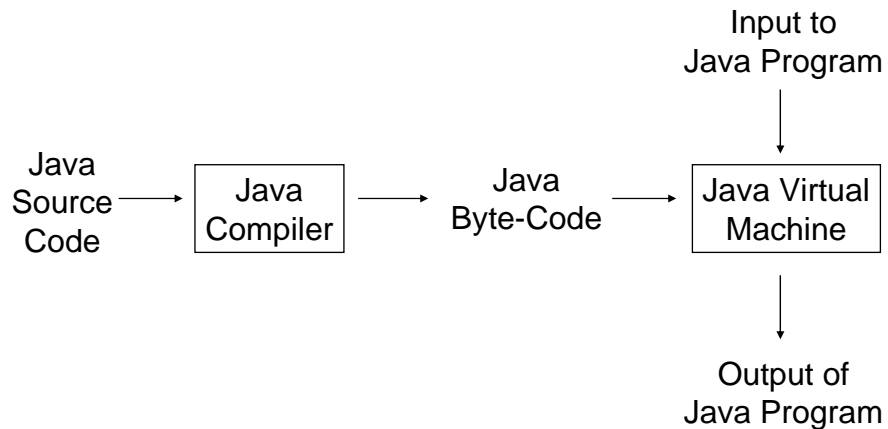
Programming Languages

- Needed to write software
- High-level languages (e.g., Java)
 - relatively easy for people to write and understand
 - not directly understood by computers
- Low-level languages (machine language)
 - directly understood by computer
 - computer-dependent

The Compiler

- A program written in a high-level language (known as the **source code**) cannot be executed directly by the computer
- A **compiler** is a program that translates source code into machine code that does the same thing (known as the **object code**)

Java Program Translation and Execution



Java Translation/Execution cont.

- Java byte-code is portable (hardware-independent)
- The Java Virtual Machine (JVM) executes Java byte-code on a real machine
- The java compiler is called **javac**
- The JVM emulator is called **java**

Algorithmic Thinking

- *Algorithm* - a set of instructions (steps) for solving a problem.
 - must be precise
 - must be complete
 - can be written in an arbitrary notation (e.g., natural language, programming language, diagram, mix, etc.)
- *Algorithmic thinking* is fundamental to computer science and programming

Example of an Algorithm

Algorithm that determines the total cost of a list of items:

1. Write the number 0 on the blackboard.
2. Do the following for each item on the list:
 - a. Add the cost of the item to the number on the blackboard.
 - b. Replace the old number on the board by this sum.
3. Announce that the answer is the number written on the board.

First Java Program

```
public class FirstProgram
{
    public static void main(String[] args)
    {
        System.out.println("Hello out there.");
        System.out.println("How's it going?");
        System.out.println(
            "Hope you are having a good day.");
        System.out.println("Good-bye.");
    }
}
```

Language Syntax

- **Syntax** of a language is a set of (grammar) rules that describe the correct way to write sentences (programs) in the language.
- Programming languages have a very precise syntax: If you break the rules, you'll get one (or more) errors.

Structure of a Java Program

```
// import needed libraries

public class ProgramName
{
    public static void main(String[] args)
    {
        // statements go here to describe
        // actions to be taken by the program
    }
}
```

A Java Statement

```
System.out.println("some message here");
```

- Outputs the message in quotes to the screen (without the quotes)

What Does FirstProgram Do?

- Take a look at the program and see if you can figure out what the program does.
- It outputs the following:

Programming Errors

- *Syntax* errors—violation of language's syntax rules, e.g., misspelling a word, forgetting a ;, etc. Caught by the compiler!
- *Runtime* errors—execution errors, e.g., division by zero.
- *Logical* errors—the program compiles and runs without runtime errors, but it does not do what it is supposed to.

Second Java Program

```
import java.util.Scanner;

public class EggBasket
{
    public static void main(String[] args)
    {
        int numberOfBaskets, eggsPerBasket, totalEggs;

        Scanner keyboard = new Scanner(System.in);

        System.out.print(
            "Enter the number of eggs in each basket: ");
        eggsPerBasket = keyboard.nextInt();
        System.out.print("Enter the number of baskets: ");
        numberOfBaskets = keyboard.nextInt();

        totalEggs = numberOfBaskets * eggsPerBasket;

        System.out.println(eggsPerBasket + " eggs per basket.");
        System.out.println(numberOfBaskets + " baskets.");
        System.out.println("Total number of eggs is " + totalEggs);
    }
}
```

What Does EggBasket Do?

- Take a look at the program and see if you can figure out what it does.

What Is a Program Variable?

```
int numberOfBaskets, eggsPerBasket, totalEggs;
```

- This is a *declaration* of three integer variables
- A **variable** is a named location to store data, i.e., a container for data
- Each variable can hold only one type of data; for example only integers, only floating point (real) numbers, or only characters
- All program variables **must** be *declared* before being used

What Is a Program Type?

- A variable's **type** determines the kind of values that a variable can hold and what operations can be applied to it.
- Some Java *primitive* types:
 - **int** (integer, whole values, e.g., 0, 1, -13, 231)
 - **double** (real values, e.g., 0.0, 3.1415, -2.72)
 - **char** (single character values, e.g., 'a', '3', '\$')
 - **boolean** (only one of two values: **true**, **false**)

How Do We Assign/Change the Value of a Variable?

```
eggsPerBasket = keyboard.nextInt();  
totalEggs = numberOfBaskets * eggsPerBasket;
```

- **Assignment statement:**
`variable = expression;`
- Assigns the value of the expression on the right side of = to the variable on the left side
- It does not mean “equal” like in math!

What Is an Expression?

```
numberOfBaskets * eggsPerBasket
```

- Program expressions are very much like arithmetic expressions you are familiar with (usual operators, parenthesis, precedence rules, etc.)
- Expressions can be evaluated to produce a value and they have a type (the type of the value of the expression)

Numeric Operators

- Some common integer operators:
 - + (obvious)
 - - (obvious)
 - * (obvious)
 - / (integer division, e.g., $6/2=3$, $5/2=2$, $19/5=?$)
 - % (mod operator, i.e., remainder of integer division, e.g., $6\%2=0$, $5\%2=1$, $19\%5=?$)
- Some common real operators:
 - +, -, *, / (real division)

Some Expressions: What Are Their Values?

```
int i = 12, j = 5, k = -3;  
double x = 2.1, y = -1.5, z = 3.0;
```

- $(i + j + k) / 3$
- $(i / j) * j + (i \% j)$
- $x * x + y * y$
- $(x + y + z) / (x - y - z)$
- $2.0 * z - (x + y)$

Output Statements

```
System.out.println(output1 + output2 + ... + outputN);  
System.out.print(output1 + output2 + ... + outputN);
```

- Concatenates the various outputs (quoted strings, variables, constants/numbers) and prints them to the screen (*println* adds a newline).
- What do the following statements output?

```
int day = 19;  
System.out.print("January " + day + ", " + 2004 +  
    " is Martin Luther King's Day! ");
```

Input Statements

Given the declaration:

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

- Declare and input an integer value:

```
int i = in.nextInt();
```
- Declare and input a real value:

```
double x = in.nextDouble();
```
- Declare and input a whole line (a string of characters):

```
String s = in.nextLine();
```
- Input of a single character with the Scanner class is trickier; we'll see it later.

To Sum Up...

- So far, we have seen
 - how to *input* values from the keyboard
 - how to *output* messages/values to the screen
 - how to create variables to store values
 - how to store values in variables and compute new values with expressions

It's Your Turn!

- Now let's put it all together: Write a Java program called *ComputeArea*, which asks the user for the *width* and *height* of a rectangle and computes and prints the area of the rectangle.

Complete ComputeArea

The `char` Type

- A variable of the `char` data type stores a single “printable” character
- A `char` constant or literal is a character in single quotes, e.g., `'y'`
- For example:

```
char answer = 'y';  
System.out.println(answer);  
prints (displays) the letter y
```

The String Type

- A string is a sequence of characters
- The String type is used to declare variables that store strings
- The String type is not a *primitive* type: it is known as a *class* or *reference* type
- A String constant is one or more characters in double quotes, e.g., "string constant"
- Examples:

```
char charVariable = 'a'; //single quotes
String stringVariable = "a"; //double quotes
String sentence = "Hello, world";
```

String Variables

- Declare a String variable:

```
String greeting;
```
- Assign a value to the variable:

```
greeting = "Hello!";
```
- Use the variable as a String argument in a *method* call:

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

causes the string Hello! to be displayed on the screen

Indexing Characters Within a String

- The index of a character within a string is an integer starting at 0 for the first character and gives the position of the character
- For example:

```
String str = "This is a string";
```

T	h	i	s		i	s		a		s	t	r	i	n	g
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Methods

- A *method* is an operation with a name and a list of arguments (possibly empty)
- A method can simply perform an action or it can return a value
- A call to a method that does not return a value is a *statement*, e.g.,
`System.out.println("Hi there!");`
- A call to a method that returns a value is an *expression*, e.g.,
`int i = keyboard.nextInt();`

Some String Methods

- The String type has many methods that allow us to manipulate String values/variables
- String methods are called/invoked with the following syntax:

```
stringVar.methodName( arguments )
```

Some String Methods

- **int** length()
returns the number of characters in the given string, e.g.,

```
String str = "This is a string";  
System.out.println(str.length());
```

What's the output?

String Methods cont.

- **char** `charAt(int pos)`
returns the character at position *pos* in the given string, e.g.,

```
String str = "This is a string";  
System.out.println(str.charAt(0));  
System.out.println(str.charAt(1));  
System.out.println(str.charAt(15));
```

What's the output?

Some String Methods cont.

- **String** `substring(int start, int end)`
returns the string starting at position *start* and ending at position (*end-1*) in the given string, e.g.,

```
String str = "This is a string";  
System.out.println(str.substring(0,4));  
System.out.println(str.substring(5,7));  
System.out.println(str.substring(0,16));
```

What's the output?

Some String Methods cont.

- `int indexOf(String aString)`
returns the position of the first occurrence of string *aString* in the given string (or -1 if not found), e.g.,

```
String str = "This is a string";  
System.out.println(str.indexOf("This"));  
System.out.println(str.indexOf("is"));  
System.out.println(str.indexOf("yoh"));
```

What's the output?

Concatenating (Appending) Strings

- As we have seen before the `+` operator can be used to concatenate string values, e.g.,

```
String name = "Cindy";  
String greeting = "Hi, " + name + "!";  
System.out.println(greeting);
```

What is the output?

Single Character Input

Given the import:

```
import java.util.Scanner;
```

and the declaration:

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

- Declare and input a single character:

```
String s = in.nextLine();
```

```
char c = s.charAt(0);
```

- Note: actually, input a whole line.

Escape Characters

- How do you print the following string?

```
The word "hard"
```

- Would this do it?

```
System.out.println("The word "hard");
```

- No, it would give a compiler error - it sees the string
The word between the first set of double quotes
and is confused by what comes after

- Use the backslash character, "\", to escape the special meaning of the internal double quotes:

```
System.out.println("The word \"hard\");
```

String Program

```
public class StringTest
{
    public static void main(String[] args)
    {
        String greeting = "Hello!";
        int len = greeting.length();
        System.out.println("Length is " + len);
        char ch = greeting.charAt(3);
        System.out.println("Character at position 3 is " + ch);
        String sub = greeting.substring(1, 3);
        System.out.println("Substring[1..3] is " + sub);
        int index1 = greeting.indexOf("lo");
        System.out.println("Index of \"lo\" is " + index1);
        int index2 = greeting.indexOf("low");
        System.out.println("Index of \"low\" is " + index2);
    }
}
```

What Is The Output Of StringTest?

- Trace through the statements of StringTest and determine the output produced by the program.

Your Turn, Again!

- Write a Java program called *BreakName*, which asks the user for his/her name in the form *First M. Last*, and outputs First, M., and Last on three different lines.
- In other words, after the name is read from input, the program needs to break it up in the three pieces (First, M., and Last) and output those one line at a time.

BreakName

Documentation and Style

- Use meaningful names for variables, programs, etc.
- Use indentation and line spacing as shown in the examples in the text
- Always include a “prologue” (a brief explanation of the program at the beginning of the file)
- Use all lower case for variables, except capitalize internal words (`eggsPerBasket`)

Comments

- *Comment*—text in a program that the compiler ignores
- Does not change what the program does, only explains the program
- Write meaningful and useful comments
- Comment the *non-obvious*
- Assume a *reasonably* knowledgeable reader
- `//` for single-line comments
- `/* ... */` for multi-line comments

Flow of Control

- The order in which statements in a program are executed is called the *flow of control*
- So far we have only seen sequential execution: statements execute one after the other in the order in which they appear in the program

Flow of Control cont.

- Consider the following tasks:
 - You want to compute the quotient of two variables but only if the divisor is not zero
 - You input some value (e.g., a date) and if it is in the correct format (mm/dd/yyyy) you continue the computation, otherwise you print an error
 - Given a grade between 0 and 100, you want to convert the numeric value to a letter grade (e.g., A for grade greater than 90, B for grade between 80 and 90, etc.)
- How can we check these conditions and execute the appropriate piece of code depending on the outcome of the check?

Selection Statements

```
Scanner in = new Scanner(System.in);
System.out.print("Enter the dividend: ");
int dividend = in.nextInt();
System.out.print("Enter the (non-zero) divisor: ");
int divisor = in.nextInt();
if (divisor != 0) {
    int quotient = dividend / divisor;
    System.out.println(
        dividend + " / " + divisor + " = " + quotient);
}
else {
    System.out.println("Cannot divide by 0");
}
```

The boolean Type

- A variable of the **boolean** data type stores one of two values: **true** or **false**
- **true** and **false** are the only two boolean constants
- boolean values/expressions are used to make decisions in programs
- For example:

```
if (x > 0) // boolean expression
{
    System.out.println("x is positive");
}
```

Boolean Expressions

- There are several kinds of boolean-valued expressions:

- a boolean variable or constant, e.g.,

```
boolean boolVar = in.nextBoolean();  
if (boolVar) { ... }
```

- an arithmetic expression followed by a relational operator followed by an arithmetic expression, e.g.,

```
int intVar = in.nextInt();  
if (intVar > 0) { ... }
```

Relational Operators

- | | |
|------------------|----------|
| ▪ == (equal) | $x == y$ |
| ▪ != (not equal) | $x != y$ |
| ▪ > | $x > y$ |
| ▪ < | $x < y$ |
| ▪ >= | $x >= y$ |
| ▪ <= | $x <= y$ |

Boolean Operators

- We can also build boolean expressions by combining two boolean expressions with a boolean operator:
 - `&&` (and) `(x > 0) && (x < 10)`
 - `||` (or) `(x <= 0) || (x >= 10)`
 - `!` (not) `!(x == 0)`

Boolean Operators cont.

- If A and B are boolean expressions, **A && B** is true if and only if both A and B are true (in other words, if either A or B or both are false, A && B is false)
- If A and B are boolean expressions, **A || B** is true if either A or B or both are true (in other words, A || B is false only if both A and B are false)
- If A is a boolean expression, **!A** is true if A is false, and **!A** is false if A is true

Some Boolean Expressions

- What's the value of each of the following expressions:

```
int x = 5, y = 12;
```

```
boolean a = true, b = false, c = true;
```

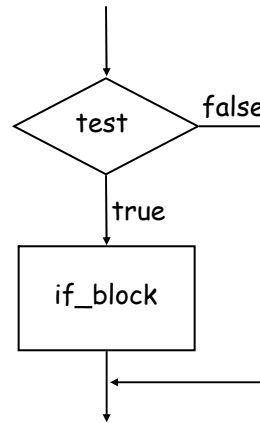
- `(x > 0) && (x < 10)`
- `(x <= 0) || (x >= 10)`
- `!(a && b && c)`
- `(a || b || c)`
- `((x - 1) == ((y / 5) + (y % 5)))`
- `((x != y) || !(x == y))`

Your Turn

- Given three integer variable, *i*, *j*, and *k*, write a boolean expression for each of the following problems:
 - *i* is equal to 3 or 5
 - *i* is between 1 and 7 (but not 1 or 7)
 - *i* is even
 - *i* is odd
 - *i* is the smallest of *i*, *j*, and *k*

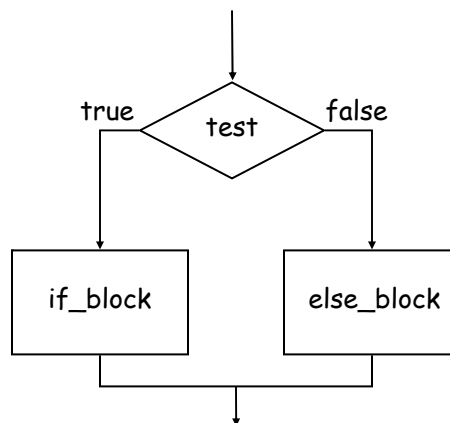
If: Syntax and Flow Chart

```
if ( test )  
{  
    if_block  
}
```



If-Else: Syntax and Flow Chart

```
if ( test )  
{  
    if_block  
}  
else  
{  
    else_block  
}
```



An Example

- Given an integer i , write a piece of code that outputs “even” or “odd” depending on whether the value of i is even or odd.

Another Example

- Given two integers i and j , write a piece of code that sets integer variable max to the value of the larger of the two.

Your Turn

- Given three integers i , j , and k , write a piece of code that sets integer variable max to the value of the largest of the three.

Max Of Three

Your Turn, Again

- Given an integer, *grade*, holding a grade between 0 and 100, write a piece of code that converts the numeric value to a letter grade according to the following table and prints the letter grade.

$grade \geq 90$	A
$80 \leq grade < 90$	B
$70 \leq grade < 80$	C
$60 \leq grade < 70$	D
$grade < 60$	E

Grade Conversion

Your Turn, One More Time

- Given a String, *s*, which is meant to hold a date in the **mm/dd/yyyy** format, check that it is in the correct format and print an error message if it is not.
- **boolean** `Character.isDigit(char ch)` allows you to check if a given character, *ch*, is a digit ('0', '1', '2', '3', '4', '5', '6', '7', '8', '9') or not.

Check Date Format

Flow of Control: Part 2

- Recall that the order in which statements in a program are executed is called the *flow of control*
- So far we have seen *sequential* execution (statements execute one after the other in the order in which they appear in the program) and *branching/selection* control structures/statements (statements are executed conditionally).

Flow of Control: Part 2 cont.

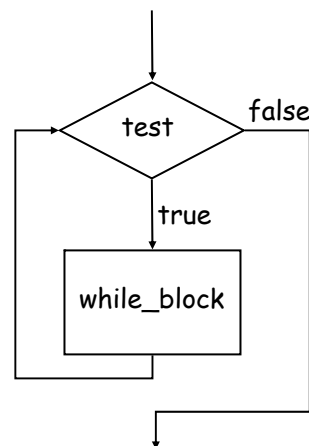
- Consider the following tasks:
 - You want to input a non-zero integer value and want to ensure that the program does not continue until the user has entered a proper value.
 - You input a sequence of values of arbitrary length, and you want to count the number of values and compute the average.
 - Given a string value, you want to find in it *all* occurrences of some given substring.
- How can we write code to perform any of these tasks involving repetition?

Iterative Statements (Loops)

```
Scanner in = new Scanner(System.in);
System.out.print(
    "Enter a non-zero integer: ");
int nonZeroInt = in.nextInt();
while (nonZeroInt == 0) {
    System.out.println(
        "Error: non-zero integer expected");
    System.out.print(
        "Enter a non-zero integer: ");
    nonZeroInt = in.nextInt();
}
```

While: Syntax and Flow Chart

```
while ( test )
{
    while_block
}
```



An Example

- Write a program segment that reads a sequence of integers until a 0 is encountered, and computes and outputs the sum of the numbers read.

Your Turn

- Write a program segment that reads a sequence of integers until a 0 is encountered, and counts and outputs the number of even and the number of odd numbers read (not including the final 0).

Count Odd/Even

Your Turn, Again

- Write a program segment that given a String variable *str* and a character variable *ch*, counts and outputs the number of occurrences of character *ch* in String *str*.

Count Character

Your Turn, One More Time

- Write a program segment that given two integer variables, *width* and *height*, outputs a rectangle of '+'s of the given width and height.
- Start by solving a simpler problem: given integer variable *width*, output *width* '+'s on one line.

Output One Row Of '+'s

- How are we going to output *height* rows of *width* columns of '+'? In other words, how do we repeat the code above *height* times?

Output A Rectangle Of '+'s

Organizing Programs

- Consider the following issues:
 - As programs increase in complexity and size, how can we break them up into more manageable pieces?
 - How can we avoid duplicating code to perform the same action in various places in our programs?

Methods

- One of the mechanism provided by Java to organize program structure is *static/class* methods
- Informally, a method is a sequence of statements that performs some task. These statements are grouped together and given a name (the name of the method)

A Simple Example

- Let's consider a program that, after asking the user for two positive numbers, *width* and *height*, outputs a rectangle of '+'s of the given width and height.

Anatomy of a Method

```
private static void procedureName(parameters)
{
    // sequence of statements
}

private static returnType functionName(parameters)
{
    // sequence of statements
    return value;
}
```

A Method (Procedure)

```
private static void outputOneRow(int w)
{
    int column = 0;
    while (column < w)
    {
        System.out.print('+');
        column = column + 1;
    }
    System.out.println();
}
```

Another Method (Function)

```
private static int inputWidth(Scanner in)
{
    System.out.print("Enter width > 0: ");
    int width = in.nextInt();
    return width;
}
```

Parameters

- We need a mechanism to provide a method with the information it needs to perform its task, e.g.,
 - What will a method to compute the area of a rectangle need?
 - What will a method to print the average of two integers need?
 - What will a method to count the number of occurrences of a character in a string need?

Parameters cont.

- In a method declaration, we specify the *formal parameter list*, which looks like a list of variable declarations separated by commas, e.g.,
`int a, int b, double c, char d, String e`
- You can have 0 or more parameters for your methods and they can be of any data type
- Choose meaningful names for the formal parameters, just like you would for variables

Examples of Method Headers

- `private static double area(
 double width, double height)`
- `private static void printAverage(
 int x, int y)`
- `private static int occurrences(
 char ch, String str)`

Examples of Method Calls

```
Scanner keyboard = new Scanner(System.in);  
double w = keyboard.nextDouble();  
double h = keyboard.nextDouble();  
double myArea = area(w, h);  
  
int i = 21, j = 13;  
printAverage(i, j);  
  
String s = "This couldn't be more fun!";  
int count = occurrences('u', s);
```

What Happens?

```
import java.util.Scanner;
public class ComputeArea
{
    public static void main(String[] args)
    {
        Scanner keyboard = new Scanner(System.in);
        double w = keyboard.nextDouble();
        double h = keyboard.nextDouble();
        double myArea = area(w, h);
        System.out.println(myArea);
    }
    private static double area(double width, double height)
    {
        double a = width * height;
        return a;
    }
}
```

Procedures vs. Functions

- Procedures are declared with void; functions are declared with a return type
- Procedures perform an action; functions compute a value
- Procedures do not return a value; functions must return a value
- Procedure calls are statements; function calls are expressions

Why Methods?

- Methods are needed for at least two different reasons:
 - They allow us to better structure and organize our programs by breaking up possibly large pieces of code into smaller, more manageable pieces
 - When the same code solving some task appears in multiple places in our program, we can write the code once in a method, and execute the code in multiple places simply through a method call

Your Turn

- For each of the following tasks design an appropriate method header:
 - The task is to compute the integer average of three given integer numbers
 - The task is to output to the screen the information of a doctor's patient, given the name, age, weight (in pounds), and height (in feet) of the patient
 - Given the first name, middle initial, and last name of a person, the task is to concatenate them and return the result (e.g., "Earl E. Bird")

Your Turn cont.

- Compute integer average of three integers
- Print information of patient, given name, age, weight (in pounds), height (in feet) of patient
- Concatenate first name, middle initial, and last name of a person, and return the result

Your Turn, Still

- Design and implement a class method that inputs a sequence of non-negative real numbers from a given Scanner, and returns the average of the numbers read. The method stops reading numbers when a negative number is entered.

Method: average

Your Turn, Last Time

- Design and implement a class method that given a `String` and a character, returns the index of the last occurrence of the character in the string (or `-1` if the character does not occur in the string).

Method: indexOfLast

A New Problem

- Consider the following task:
 - Input N real numbers representing temperature measurements and compute the following:
 - the average, minimum, and maximum temperature
 - the number of temperatures that are above the average and the number of temperatures that are below the average
 - the median temperature

What's the Problem?

- We need to store *all* the temperatures *before* we can perform some of the computation
- How many variables do we need?
- What are we going to name them?

The Solution: Arrays

- An *array* is a sequence of variables of the same data type (any type can be used: e.g., int, double, boolean, String, etc.)
- Each variable in the array is called an *element*
- Array elements are accessed through an *index* (their position in the array)

An Example

```
int N = 5;
double [] temperatures;
temperatures = new double[N];

Scanner in = new Scanner(System.in);
int pos = 0;
while (pos < N)
{
    temperatures[pos] = in.nextDouble();
    pos = pos + 1;
}
```

Declaring Arrays

- Arrays are declared like normal variables with the addition of [] between type and name, e.g.

```
double [] temperatures;
int [] scores;
boolean [] answers;
String [] names;
```
- The declaration does not specify the number of elements
- Declaring an array **does not** reserve (allocate) memory for the array elements

Allocating The Array

- To allocate memory space for the array elements we need to use the **new** operator, e.g.,

```
temperatures = new double[10];  
int numScores = 8;  
scores = new int[numScores];  
answers = new boolean[4];  
names = new String[5];
```
- **new** is followed by the type of the elements and the number of elements between []
- The number of elements can be any expression that evaluates to a positive integer value

Accessing Array Elements

- Elements of an array are accessed by using the name of the array variable followed by the index (position in the array) of the element between [], e.g.,

```
temperatures[2]  
scores[numScores-1]  
answers[0]  
names[3]
```
- The index can range between 0 and the number of elements in the array - 1

Array Length

- Once we declare and allocate an array, we can retrieve the array length (the number of elements in the array) from a variable called *arrayName.length*, e.g.,

`temperatures.length` is 10

`scores.length` is 8

`answers.length` is 4

`names.length` is 5

Your Turn

- Declare and allocate an array of 10 integers, call it *a*

Your Turn cont.

- Write a loop that initializes the 10 elements of the array to the integers 1, 2, 3, 4, 5, 6, 7, 8, 9, 10

Your Turn cont.

- Write a loop that outputs the elements of the array, one per line

Your Turn cont.

- Write a loop that outputs the elements of the array, one per line, **in reverse order**

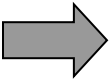
Your Turn cont.

- Write a piece of code that checks if the elements in an array of char form a palindrome

For Loops

- A different syntax for a familiar concept
- They work well with arrays

```
for (init; test; update)
{
    for_block
}
```



```
init
while (test)
{
    for_block
    update
}
```

Example

- Rewrite the loop that initializes the 10 elements of an array to the integers 1, 2, 3, 4, 5, 6, 7, 8, 9, 10

```
int i = 0;
while (i < 10)
{
    a[i] = i + 1;
    i++;
}
```

Your Turn (Our Old Friend)

- Write a for loop that given a String variable *str* and a character variable *ch*, counts and outputs the number of occurrences of character *ch* in String *str*.

Count Character

Your Turn, Again

- Write a program segment—using for loops—that given two integer variables, *width* and *height*, outputs a rectangle of '+'s of the given width and height.

Output A Rectangle Of '+'s

Arrays And Methods

- Arrays can be passed as parameters to methods and can be returned by methods (functions), e.g.,

```
▪ private static void printArray(double[] a)
▪ private static void clearArray(int[] a)
▪ private static int[] copyArray(int[] a)
▪ private static int indexOfSmallest(
    int startIndex, double[] t)
▪ private static void sort(double[] temps)
```

Print Array

- Given an array of double, output each element on a separate line (in the order in which they occur in the array)

```
private static void printArray(double[] a)
{

}
}
```

Clear Array

- Given an array of int, set each element of the array to 0

```
private static void clearArray(int[] a)
{

}
}
```

Copy Array

- Given an array of int, return a new array which is a copy of the given one (same length, same elements)

```
private static int[] copyArray(int[] a)
{

}
}
```

FileIOHelper and Student

- The *FileIOHelper* class provides two class methods:
 - `int FileIOHelper.getNumberOfStudents(String fileName)`: takes a **String** parameter which is the name of an input file in the format described in the lab, and returns the number of students in the file;
 - `Student FileIOHelper.getNextStudent()`: takes no parameters, inputs the name and scores for the next student in the input file, and returns a *Student* object with the corresponding value.

FileIOHelper and Student cont.

- A *Student* object has a name (**String**) and three scores (**integers**). The *Student* class provides the following (instance) methods to manipulate *Student* objects:
 - `Student(String name)`: returns a *Student* object with the given name and 0 for all the scores (constructor)
 - `String std.getName()`: returns the name of student *std*
 - `int std.getScore(int i)`: returns the *i*-th score (*i* between 1 and 3) for student *std*
 - `void std.setScore(int i, int x)`: sets the *i*-th score (*i* between 1 and 3) for student *std* to the value *x* ($x \geq 0$)

FileIOHelper and Student cont.

```
int n = FileIOHelper.getNumberOfStudents("info.txt");
System.out.println("Number of students: " + n);

Student std = FileIOHelper.getNextStudent();
System.out.println("Name: " + std.getName());

int score1 = std.getScore(1);
std.setScore(2, score1 + 20);

System.out.println("Score #1: " + score1);
System.out.println("Score #2: " + std.getScore(2));
System.out.println("Score #3: " + std.getScore(3));
```

Java Classes

- Java programs are made up of *classes*
- So far, we have *written* only the “main program” class
- But we have *used* several other classes:
 - Scanner
 - String
 - FileIOHelper
 - Student

Two Kinds of Classes

- Classes can either provide a new *type* with corresponding *methods* to manipulate the values of that type, e.g.,
 - Scanner
 - String
 - Student
- Or they simply can provide a bunch of methods that perform some useful task, e.g.,
 - FileIOHelper

Two Kinds of Methods

- You may have noticed a difference in the syntax of method invocation
- We write
 - `str.length()`
 - `in.nextInt()`
- But we also write
 - `FileIOHelper.numberOfStudents(fn)`
 - `FileIOHelper.getNextStudent()`

Sending Output to a (Text) File

```
import java.util.Scanner;
import java.io.*;
public class TextFileOutputDemol
{
    public static void main(String[] args) throws IOException
    {
        Scanner in = new Scanner(System.in);
        System.out.print("Enter your name: ");
        String name = in.nextLine();
        PrintWriter outputFile =
            new PrintWriter(new FileWriter("out.txt"));
        outputFile.println(name);
        outputFile.close();
    }
}
```

Creating a File for Text Output

- We need a couple of new classes:
 - `PrintWriter`
 - `FileWriter`
- To gain access to them we need to import them with:
 - `import java.io.*;`
- To create and open the file:
 - `PrintWriter outFile =`
`new PrintWriter(`
`new FileWriter(fileName));`

Writing Output to a Text File

- The `PrintWriter` class has output methods you are already familiar with:
 - `print`
 - `println`
- Instead of calling `System.out.print(...)` or `System.out.println(...)`, you invoke:
 - `outFile.print(...)`
 - `outFile.println(...)`

Closing the File

- It is important to remember to explicitly close a file you are done writing to:
 - `outFile.close()`

Your Turn

- Write a program that asks the user for a file name and then outputs the numbers 1 through 10, one per line, to the corresponding file.

Output Numbers

What If an Error Occurs?

- What could go wrong when we try to create a file?
- We might not be able to create it!
- In that case an *exception* (error) is generated/raised by the program.
- What can we do about it?

Exceptions

- There are a variety of errors that can occur in a Java program that result in an exception being raised
- Java forces us to deal with some of them by
 - either explicitly stating that such an exception might occur in our program
 - or by catching the exception and dealing with it in our code

Stating That Exception Might Occur

```
import java.util.Scanner;
import java.io.*;
public class TextFileOutputDemo1
{
    public static void main(String[] args) throws IOException
    {
        Scanner in = new Scanner(System.in);
        System.out.print("Enter your name: ");
        String name = in.nextLine();
        PrintWriter outputFile =
            new PrintWriter(new FileWriter("out.txt"));
        outputFile.println(name);
        outputFile.close();
    }
}
```

Dealing With Exception If It Occurs

```
import java.util.Scanner;
import java.io.*;
public class TextFileOutputDemo2
{
    public static void main(String[] args)
    {
        Scanner in = new Scanner(System.in);
        System.out.print("Enter your name: ");
        String name = in.nextLine();
        try {
            PrintWriter outputFile =
                new PrintWriter(new FileWriter("out.txt"));
            outputFile.println(name);
            outputFile.close();
        } catch (IOException e) {
            System.out.println("Error opening the file out.txt");
        }
    }
}
```

Try-Catch

- You can think of it as a control structure in that it affects the flow of execution of the program when an exception occurs.

```
try {
    // statements possibly raising exception
}
catch (exception declaration)
{
    // statements dealing with exception
}
```

Reading Input from a (Text) File

```
import java.util.Scanner;
import java.io.*;

public class TextFileInputDemo1
{
    public static void main(String[] args)
        throws IOException
    {
        File file = new File("in.txt");
        Scanner inputFile = new Scanner(file);
        String line = inputFile.nextLine();
        inputFile.close();
        System.out.println(line);
    }
}
```

Opening a File for Text Input

- We need a new class:
 - File
- To gain access to it we need to import it with:
 - `import java.io.*;`
- To open an existing file:
 - `File file = new File(fileName);`
`Scanner inFile = new Scanner(file);`

Reading Input from a Text File

- The Scanner class has input methods you are already familiar with:
 - `nextLine`, `nextInt`, `nextDouble`, etc.
- Now we also need a way to check when we reach the end of the file:
 - `boolean hasNext()`
 - It takes no parameters and returns true if there is more data to read, and false otherwise

Closing the File

- It is important to remember to explicitly close a file you are done reading from:
 - `inFile.close()`

Your Turn

- Complete the following program that asks the user for a file name, opens the file, reads one line at a time and outputs the line to the screen, until it reaches the end of the file.
- Make sure you catch the possible `IOException` in a try-catch statement.

Average Numbers

```
import java.util.Scanner;
import java.io.*;

public class AverageNumbers
{
    public static void main(String[] args)
    {

    }
}
```

Passing Arguments To Main

```
public static void main(String[] args) {...}
```

- The formal parameter `args` is an array of `String`
- How do we pass actual arguments to method `main`?

Command Line Arguments

- When you invoke a Java program from a terminal, anything you type after the name of the program is passed to the main method as the array of String, e.g.,

```
java SomeProgram arg1 arg2 arg3
```

Example

```
public class CommandLineArgsTest
{
    public static void main(String[] args)
    {
        System.out.println(
            "Number of arguments: " + args.length);
        for (int i = 0; i < args.length; i++)
        {
            System.out.println(args[i]);
        }
    }
}
```

Your Turn

- What will the following program invocations output to the screen?
 - `java CommandLineArgsTest this is a test`
 - `java CommandLineArgsTest 1 2 3 4 5`
 - `java CommandLineArgsTest to b || ! 2 be`
 - `java CommandLineArgsTest`

```

/*
 * Rectangle1.java
 *
 * Created on Feb 1, 2004
 * Updated on Apr 25, 2005
 *
 * Author: Paolo Bucci
 *
 * This program outputs a rectangle of '+'s of width and height
 * provided by the user. (The whole program is in the main method.)
 */

import java.util.Scanner;

public class Rectangle1
{
    public static void main(String[] args)
    {
        Scanner keyboard = new Scanner(System.in);

        // Input the width of the rectangle
        System.out.print("Enter width > 0: ");
        int width = keyboard.nextInt();

        // Input the height of the rectangle
        System.out.print("Enter height > 0: ");
        int height = keyboard.nextInt();

        // Output height rows of width columns of '+'s
        int row = 0;
        while (row < height)
        {
            // Output one row of width '+'s
            int column = 0;
            while (column < width)
            {
                System.out.print('+');
                column = column + 1; // go to the next column
            }
            System.out.println(); // terminate the row

            row = row + 1; // go to the next row
        }
    }
}

```

```

/*
 * Rectangle2.java
 *
 * Created on Feb 1, 2004
 * Updated on Apr 25, 2005
 *
 * Author: Paolo Bucci
 *
 * This program outputs a rectangle of '+'s of width and height
 * provided by the user. (The program is organized with methods.)
 */

import java.util.Scanner;

public class Rectangle2
{
    public static void main(String[] args)
    {
        Scanner keyboard = new Scanner(System.in);

        int width = inputWidth(keyboard);
        int height = inputHeight(keyboard);
        outputRectangle(width, height);
    }

    private static int inputWidth(Scanner in)
    {
        // Input the width of the rectangle
        System.out.print("Enter width > 0: ");
        int width = in.nextInt();
        return width;
    }

    private static int inputHeight(Scanner in)
    {
        // Input the height of the rectangle
        System.out.print("Enter height > 0: ");
        int height = in.nextInt();
        return height;
    }

    private static void outputRectangle(int w, int h)
    {
        // Output height rows of width columns of '+'s
        int row = 0;
        while (row < h)
        {
            outputOneRow(w);
            row = row + 1; // go to the next row
        }
    }

    private static void outputOneRow(int w)
    {
        // Output one row of width '+'s
        int column = 0;
        while (column < w)
        {
            System.out.print('+');

```

```
        column = column + 1; // go to the next column
    }
    System.out.println(); // terminate the row
}
}
```

```

/*
 * Rectangle3.java
 *
 * Created on Feb 1, 2004
 * Updated on Apr 25, 2005
 *
 * Author: Paolo Bucci
 *
 * This program outputs a rectangle of '+'s of width and height
 * provided by the user. (The program is organized with methods.)
 *
 */

import java.util.Scanner;

public class Rectangle3
{
    public static void main(String[] args)
    {
        Scanner keyboard = new Scanner(System.in);

        // Input the width of the rectangle
        int width = inputPositiveInteger(keyboard, "width");

        // Input the height of the rectangle
        int height = inputPositiveInteger(keyboard, "height");

        outputRectangle(width, height);
    }

    private static void outputRectangle(int w, int h)
    {
        // Output height rows of width columns of '+'s
        int row = 0;
        while (row < h)
        {
            outputOneRow(w);
            row = row + 1; // go to the next row
        }
    }

    private static void outputOneRow(int w)
    {
        // Output one row of width '+'s
        int column = 0;
        while (column < w)
        {
            System.out.print('+');
            column = column + 1; // go to the next column
        }
        System.out.println(); // terminate the row
    }

    private static int inputPositiveInteger(Scanner in, String name)
    {
        // Input a positive integer
        System.out.print("Enter " + name + " > 0: ");
        int i = in.nextInt();

        // Make sure the number entered by the user is positive
    }
}

```

```
    while (i <= 0)
    {
        System.out.println("Error: positive integer expected");
        System.out.print("Enter " + name + " > 0: ");
        i = in.nextInt();
    }

    return i;
}
}
```

```

/*
 * Temperatures1.java
 *
 * Created on Feb 11, 2004
 * Updated on May 2, 2005
 *
 * Author: Paolo Bucci
 *
 * Example of use of arrays. Reads a bunch of real numbers
 * (temperatures) and computes the average, the median, and
 * the number of temperatures above and below the average.
 */

import java.util.Scanner;

public class Temperatures1
{
    public static void main(String[] args)
    {
        Scanner keyboard = new Scanner(System.in);

        int N = keyboard.nextInt(); // input number of temperatures
        double [] temperatures = new double[N];
        double sum = 0.0;

        // input the temperatures into the array and compute the sum
        int i = 0;
        while (i < N)
        {
            temperatures[i] = keyboard.nextDouble();
            sum = sum + temperatures[i];
            i++;
        }

        // compute the average
        double average = sum / N;

        // count the number of temperatures above and below the average
        int below = 0, above = 0;
        i = 0;
        while (i < N)
        {
            if (temperatures[i] < average)
            {
                below++;
            }
            else if (temperatures[i] > average)
            {
                above++;
            }
            i++;
        }

        // order the temperatures in increasing order
        sort(temperatures);

        // compute the median of the temperatures
        double median;
        if (N % 2 == 0) // N is even
        {

```

```

        median = (temperatures[N/2-1] + temperatures[N/2]) / 2.0;
    }
    else // N is odd
    {
        median = temperatures[N/2];
    }

    // output the results
    System.out.println("average = " + average);
    System.out.println("above = " + above);
    System.out.println("below = " + below);
    System.out.println("median = " + median);
}

/*
 * Sort the temperatures array in increasing order.
 */
private static void sort(double[] temps)
{
    int index = 0;
    while (index < temps.length)
    {
        int indexOfNextSmallest = indexOfSmallest(index, temps);
        interchange(index, indexOfNextSmallest, temps);
        index++;
    }
}

/*
 * Find and return the index of the smallest value in array t
 * starting at index startIndex.
 */
private static int indexOfSmallest(int startIndex, double[] t)
{
    double min = t[startIndex];
    int indexOfMin = startIndex;
    int index = startIndex + 1;
    while (index < t.length)
    {
        if (t[index] < min)
        {
            min = t[index];
            indexOfMin = index;
        }
        index++;
    }
    return indexOfMin;
}

/*
 * Swaps the elements at positions i and j in array t.
 */
private static void interchange(int i, int j, double[] t)
{
    double tmp = t[i];
    t[i] = t[j];
    t[j] = tmp;
}
}

```

```

/*
 * Temperatures2.java
 *
 * Created on Feb 11, 2004
 * Updated on May 2, 2005
 *
 * Author: Paolo Bucci
 *
 * Example of use of arrays. Reads a bunch of real numbers
 * (temperatures) and computes the average, the median, and
 * the number of temperatures above and below the average.
 */

import java.util.Scanner;

public class Temperatures2
{
    public static void main(String[] args)
    {
        Scanner keyboard = new Scanner(System.in);

        int N = keyboard.nextInt(); // input number of temperatures
        double [] temperatures = new double[N];
        double sum = 0.0;

        // input the temperatures into the array and compute the sum
        for (int i = 0; i < N; i++)
        {
            temperatures[i] = keyboard.nextDouble();
            sum = sum + temperatures[i];
        }

        // compute the average
        double average = sum / N;

        // count the number of temperatures above and below the average
        int below = 0, above = 0;
        for (int i = 0; i < N; i++)
        {
            if (temperatures[i] < average)
            {
                below++;
            }
            else if (temperatures[i] > average)
            {
                above++;
            }
        }

        // order the temperatures in increasing order
        sort(temperatures);

        // compute the median of the temperatures
        double median;
        if (N % 2 == 0) // N is even
        {
            median = (temperatures[N/2-1] + temperatures[N/2]) / 2.0;
        }
        else // N is odd
        {

```

```

        median = temperatures[N/2];
    }

    // output the results
    System.out.println("average = " + average);
    System.out.println("above = " + above);
    System.out.println("below = " + below);
    System.out.println("median = " + median);
}

/*
 * Sort the temperatures array in increasing order.
 */
private static void sort(double[] temps)
{
    for (int index = 0; index < temps.length; index++)
    {
        int indexOfNextSmallest = indexOfSmallest(index, temps);
        interchange(index, indexOfNextSmallest, temps);
    }
}

/*
 * Find and return the index of the smallest value in array t
 * starting at index startIndex.
 */
private static int indexOfSmallest(int startIndex, double[] t)
{
    double min = t[startIndex];
    int indexOfMin = startIndex;
    for (int index = startIndex + 1; index < t.length; index++)
    {
        if (t[index] < min)
        {
            min = t[index];
            indexOfMin = index;
        }
    }
    return indexOfMin;
}

/*
 * Swaps the elements at positions i and j in array t.
 */
private static void interchange(int i, int j, double[] t)
{
    double tmp = t[i];
    t[i] = t[j];
    t[j] = tmp;
}
}

```

```

/*
 * ParameterPassingTest.java
 *
 * Created on May 16, 2005
 *
 * Author: Paolo Bucci
 *
 * Program to test the parameter passing mechanism in Java.
 */

public class ParameterPassingTest
{
    public static void main(String[] args)
    {
        // Passing primitive type values
        int i = 1;
        double d = -2.4;
        char c = '?';
        boolean b = true;
        testPrimitiveTypes(i, d, c, b);
        System.out.println(i + ", " + d + ", " + c + ", " + b);

        // Passing array values - test 1
        int [] a = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
        testArray1(a);
        for (int pos = 0; pos < a.length; pos++)
        {
            System.out.print(a[pos] + " ");
        }
        System.out.println();

        // Passing array values - test 2
        testArray2(a);
        for (int pos = 0; pos < a.length; pos++)
        {
            System.out.print(a[pos] + " ");
        }
        System.out.println();

        // Passing String values
        String s = "This is a test";
        testString(s);
        System.out.println(s);
    }

    /*
     * Modifies formal parameter values, but does not change
     * values of actual parameters when of a primitive type.
     */
    private static void testPrimitiveTypes(int i, double d, char c,
                                           boolean b)
    {
        // What are the values of i, d, c, and b here?
        i = 7;
        d = 3.14;
        c = 'Y';
        b = false;
        // What are the values of i, d, c, and b here?
    }
}

```

```

/*
 * Modifies array elements of formal parameter and changes
 * values of elements of actual array parameter.
 */
private static void testArray1(int[] a)
{
    // What are the values of a and of a's elements here?
    for (int pos = 0; pos < a.length; pos++)
    {
        a[pos] = pos;
    }
    // What are the values of a and of a's elements here?
}

/*
 * Modifies formal array parameter, but does not change
 * value of actual parameter when of an array type.
 */
private static void testArray2(int[] a)
{
    // What are the values of a and of a's elements here?
    a = new int[5];
    for (int pos = 0; pos < a.length; pos++)
    {
        a[pos] = 1;
    }
    // What are the values of a and of a's elements here?
}

/*
 * Modifies formal parameter value, but does not change
 * value of actual parameter.
 */
private static void testString(String s)
{
    // What is the value of s here?
    s = "new value";
    // What is the value of s here?
}
}

```