Introduction to Computational Thinking

Encryption/Decryption

Basic Arrays

Indefinite Loops:
Motivation; while/do-while Statements;

Program analysis;
Loop pattern: Sentinel input/Fencepost loops

Qiao Xiang, Qingyu Song

https://sngroup.org.cn/courses/ct-xmuf25/index.shtml

11/19/2025

This deck of slides are heavily based on cs112 at Yale University and cs101 at UCAS, respectively, by courtesy of Dr. Y. Richard Yang and Dr. Zhiwei Xu.

Roadmap: String Processing

- Access methods, e.g.,
  - `length`, `substring`, `charAt`, `indexOf`
- String generator methods (string->string), e.g.
  - `toLowerCase`, `toUpperCase`, `replace`, `split` (string->strings)
- Conversion from String, e.g.,
  - `Integer.parseInt`, `Double.parseDouble`
- Input processing, e.g.,
  - `Scanner.next`, `Scanner.nextLine`
- Output formatting: format string
  - `System.out.printf`, `String.format`
- Boolean methods, e.g.,
  - `equals`, `equalsIgnoreCase`, `startsWith`, `endsWith`, `contains`, `matches`

Recap: `char` vs. `String`

	String	char
Store	An object w/ a sequence of chars	A primitive value storing a unicode index
Methods	.length, .toUpperCase, …	No methods
==, !=	Compare reference, not value	Compare value
<, <=, >, >=	Not defined. Use compareTo(anotherString) method	Lexicographic order
.equals method	Compare content	Not defined

https://en.wikipedia.org/wiki/List_of_Unicode_characters

Outline

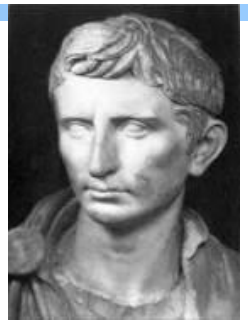
- Arrays
  - Motivation
  - Basic syntax
    - declare arrays, access array elements, OOB
    - array initializer list

Caesar Cipher

Plaintext	Ciphertext
attack said zeus	dwwdfn vdlg chxv
προσβάλλω	στςφείξξό

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c

Additional Caesar-Style Cipher

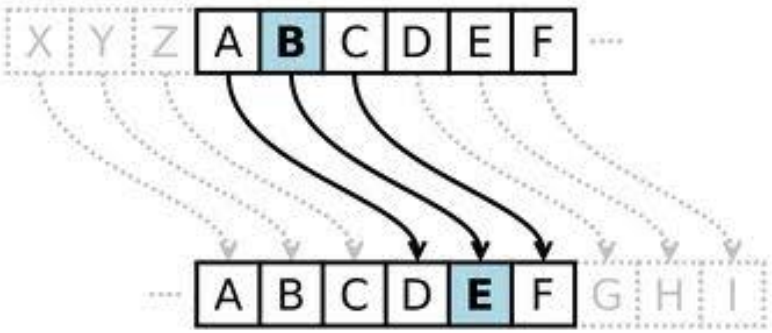


- Augustus
  - Right shift 1 without rotate, with AA for Z
- Mezuzah
  - Shift of one to encrypt the names of God

- Vigenere (a variant) was used by the Confederacy during Civil War

- Mafia boss Provenzano (2006): A (4), …

Why Programming?



Dear Cleopatra,

I love you as big as the Aegean sea, as high as Mount Olympus, as many as the stars of Hyades, Orion, Sirius, and Ursa Major combined... My soul, heart, body and kingdom are yours for the taking. When I think of you, Cleo, my heart beats like that of a gladiator facing death at the claws of a rabid tiger ...

Caesar Cipher Encoding (Design 1)

Letter	a	b	c	d	…	…	w	x	y	z
Orig	*97*	*98*	*99*	*100*	…	…	*119*	*120*	*121*	*122*
Shift (+3)	100	101	102	103			122	123	124	125
Cipher	100	101	102	103			122	97	98	99

```
int cipher = ch + key;
if (cipher > 'z') cipher -= 26;
ch  = (char)cipher;
```

Q: What if key is 100?

a b c d e f g h i j k l m n o p q r s t u v w x y z

d e f g h i j k l m n o p q r s t u v w x y z a b c

Caesar Cipher: Encoding (Design 2)

Letter	a	b	c	d	…	…	w	x	y	z
Orig	0	1	2	3	…	…	22	23	24	25
Shift (+3)	3	4	5	6			25	26	27	28
Cipher (% 26)	3	4	5	6			25	0	1	2

int cipherInt = (origInt + key) % 26

Assumption: a-z mapped to 0 to 25.

How to map a char in a-z to 0 to 25? char – ‘a’

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c

Caesar Cipher Encoding

Letter	a	b	c	d	…	…	w	x	y	z
Orig	0	1	2	3	…	…	22	23	24	25
Shift (+3)	3	4	5	6			25	26	27	28
Cipher (% 26)	3	4	5	6			25	0	1	2

```
int chRelativeIndex = ch – ‘a’;
int cipherRelInd = (chRelativeIndex + key) % 26;
ch = (char)(cipherRelInd + ‘a’);
```

a b c d e f g h i j k l m n o p q r s t u v w x y z

d e f g h i j k l m n o p q r s t u v w x y z a b c

CaesarFile using a for Loop

```
public class ReadFile {
    public static void main(String[] args) {
        try {
            Scanner input = new Scanner(new File("data.txt"));
            encode(input, 3);
        } catch (FileNotFoundException e) {
            // print an error message
            System.out.println("File not found exception");
        } // end of catch
    } // end of main
}
```

```
public static void encode(Scanner scan, int key) {

  for (int line = 0; line < NLINES; line ++ ) {
       String text = scan.nextLine();
       String result = Caesar.encode(text, key);
       System.out.println( result );
  }
}
```

Think:
How to Decode?

Caesar Cipher: Decoding

Letter	a	b	c	d	…	…	w	x	y	z
Orig	0	1	2	3	…	…	22	23	24	25
Shift (+3)	3	4	5	6			25	26	27	28
Cipher (% 26)	3	4	5	6			25	0	1	2
Back (-3)	0	1	2	3			22	-3	-2	-1
Range (+26% 26)	0	1	2	3			22	23	24	25

Unifying Design

int cipherInt = (origInt + key) % 26

int origInt = (cipherInt – key + 26) % 26

- A single encode method with parameter key
  - To encode
    - encode(key)
  - To decode
    - encode(26 - key)

Caesar.java

Unifying Design

- Encode using key

```
int chRelativeIndex = ch – ‘a’;
int cipherRelInd = (chRelativeIndex + key) % 26;
ch = (char)(cipherRelInd + ‘a’);
```

- Decode using key

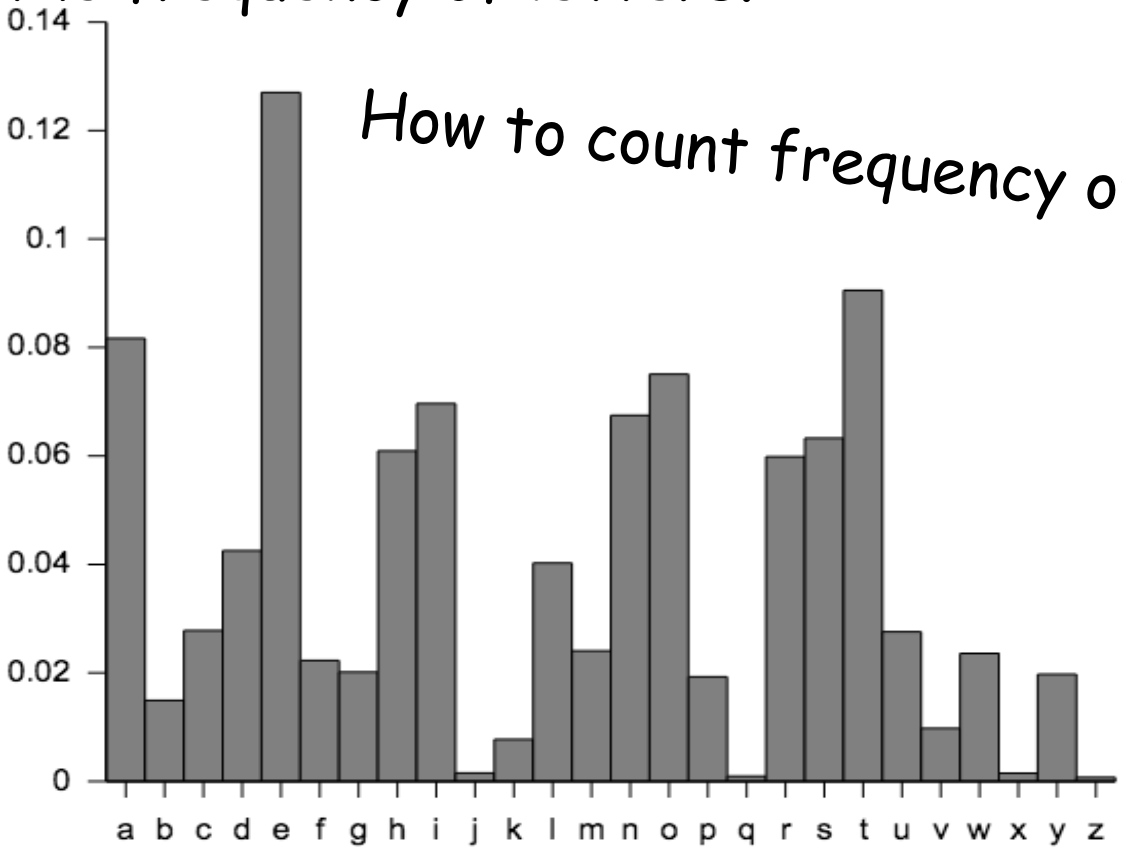
```
int chRelativeIndex = ch – ‘a’;
int cipherRelInd = (chRelativeIndex – key + 26) % 26;
ch = (char)(cipherRelInd + ‘a’);
```

decode(key) = encode(26 - key)

Caesar.java

Exercise: Breaking Caesar Encryption

- Caesar encryption (or permutation ciphers in general) is easy to break because it does not change the frequency of letters.



How to count frequency of letters?

Frequency Analysis Code (Attempt 1)

```
int nA = 0, nB = 0, nC = 0, nD = 0, nE = 0,
    nF = 0, nG = 0, … nZ = 0;

for (int line = 0; line < NLINES; line ++ ) {

       String text = scan.nextLine();

       for (int i = 0; i < text.length(); i++) {

          char ch = line.charAt( i );

          if ( ch == 'a' )
            nA ++;
          else if ( ch == 'b' )
            nB ++;
          else if ( ch == 'c' )
            nC ++;
          …
}
print nA, nB, ….
```

Improving Attempt 1

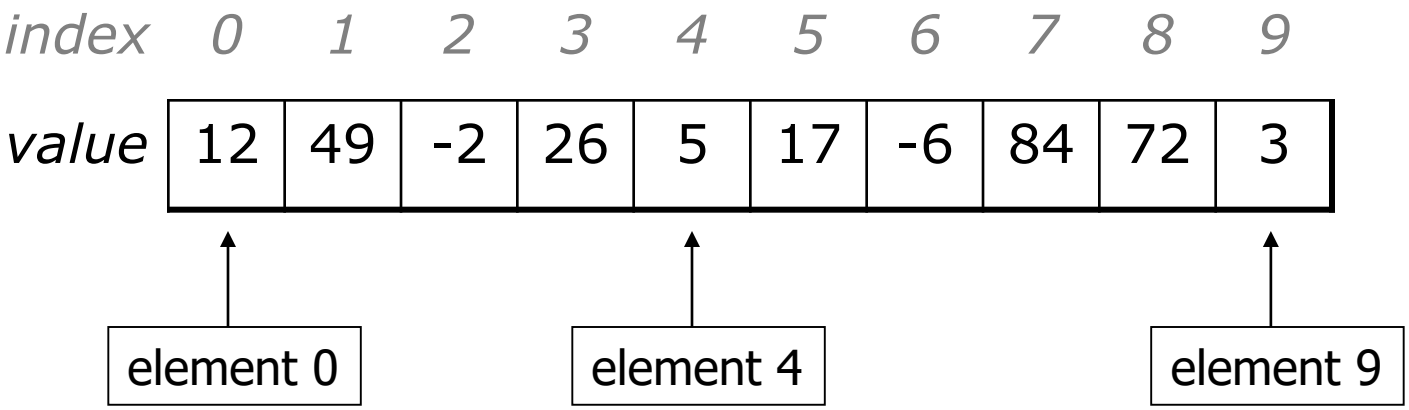
- We need a way to declare many variables in one step.
- We need an effective scheme to refer to the variables
- The objective of arrays is to provide these

Outline

- Arrays
  - Motivation
  - Basic syntax
    - declare arrays, access array elements, OOB
    - array initializer list

Arrays

- **array**: an object that stores many values of the same type.
  - **element**: one value in an array.
  - **index**: array element referred to by *position number* starting from 0



Array Declaration

After declaration, all elements of an int array are initialized to 0.

<type>[] **<name>** = new **<type>**[**<length>**];

- Example:

```
int[] numbers = new int[10];
```

index	0	1	2	3	4	5	6	7	8	9
value	0	0	0	0	0	0	0	0	0	0

- The length in declaration can be any non-negative integer expression, e.g.,

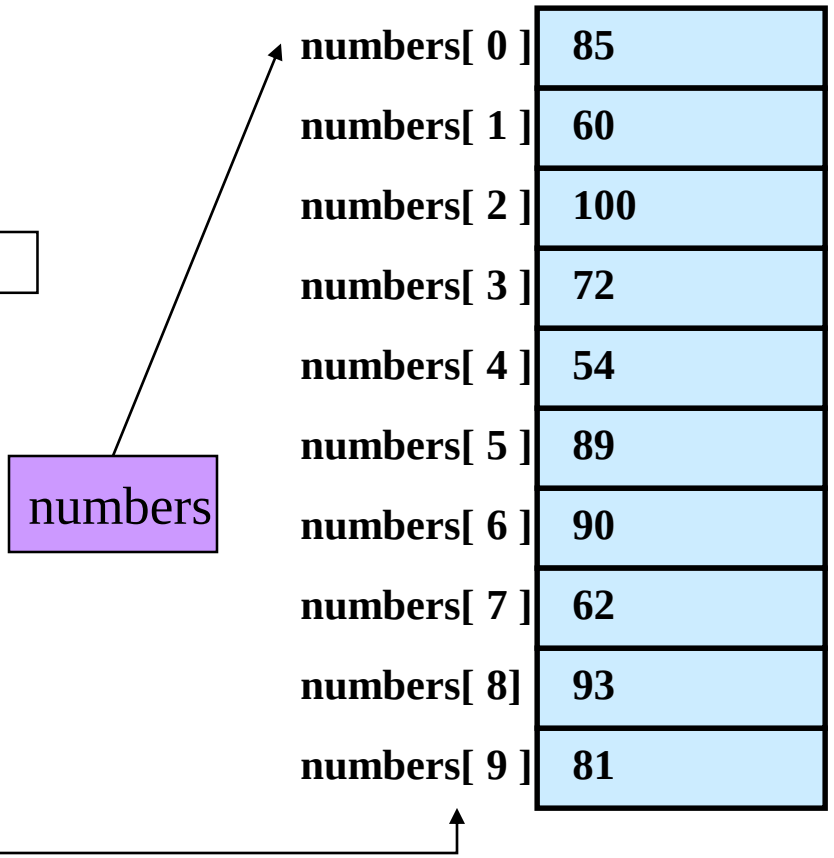
```
int m = 3;
int d = 31;
double[] data = new double[m * d +1];// arr w/ 94 elem
```

Accessing Array Elements

<name>[<index>] // access

<name>[<index>] = <value>; // modify

```
int[] numbers = new int[10];
```



position number (index or subscript) of the element within array **numbers**

BasicArrays.java

Array `length` field

- Java provides a `length` field for each array to store the number of elements
  - It does not use parentheses like a String's `.length()`
  - `length` holds the number of elements, *not* the largest index
- Reference length using the array variable's name:

  <name>`.length`

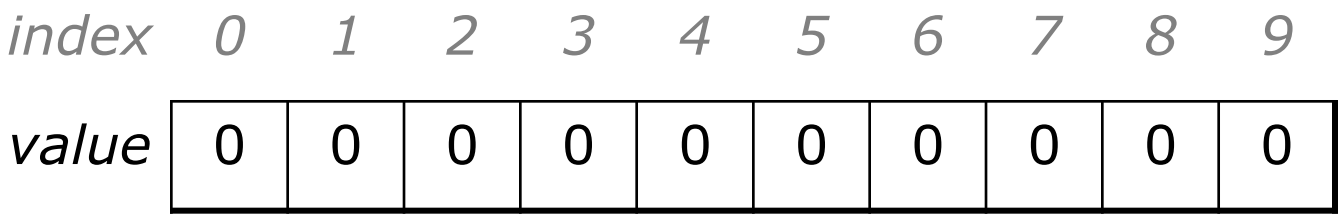
```
int[] numbers = new int[8];
System.out.println ( numbers.length ); // 8
```

- What expressions refer to the index of (by .length):
  - the last element of any array?
  - the middle element (assume odd number of elements)?

Out-of-bounds (OOB)

- Reading or writing any index outside of range will throw an `ArrayIndexOutOfBoundsException`.
- Example (which line(s) will cause OOB):

```
int[] data = new int[10];
System.out.println(data[0]);              // okay
System.out.println(data[9]);              // okay
System.out.println(data[-1]);             // error: OOB
System.out.println(data[data.length]);    // error: OOB
```



Outline

- Admin and recap
- Arrays
  - Motivation
  - Basic syntax
    - declare arrays, access array elements, OOB
    - array initializer list

Quick Array Initialization

- An initializer list can be used to instantiate and initialize an array in one step

  ***<type>*[] *<name>* = {*<value>*, *<value>*, … *<value>*};**

  - Examples:

```
int[] numbers = {12, 49, -2, 26, 5, 17, -6};
char[] letterGrades = {'A', 'B', 'C', 'D', 'F'};
String[] wordList = {“cs112“, “computer", “television"};
```

  - The values are delimited by braces and separated by commas
  - The new operator is not used
  - The compiler
    - figures out the size by counting the values in the initializer list
    - initializes the values of the elements with those in the initializer list

Outline

- Admin and recap
- Arrays
  - Motivation
  - Basic syntax
    - declare arrays, access array elements, OOB
    - array initializer list
    - array as method parameter/return

Array as Parameter (Declare)

```
public static <type> <method>(<type>[] <name>) {
```

- Example:

```
// Returns the average of the given array of numbers.
public static double max(int[] numbers) {
    int max = Integer.MIN_VALUE;
    for (int i = 0; i < numbers.length; i++) {
        if (numbers[i] > max) max = numbers[i];
    }
    return max;
}
```

- See Arrays class for many useful array methods
  - https://docs.oracle.com/javase/7/docs/api/java/util/Arrays.html

Array as Parameter (Call)

<methodName>`(`*<arrayName>*`);`

- Example:

```
public class MyProgram {
    public static void main(String[] args) {
        // figure out the average TA IQ
        int[] iq = {126, 109, 149, 167, 125};
        int maxIQ = max(iq);
        System.out.println("Highest IQ = " + maxIQ);
    }
    ...
```

  - Notice that you don't write the `[]` when passing the array.

Example: Command-Line Arguments

- The signature of the `main` method indicates that it takes an array of `String` as a parameter
- These values come from command-line arguments that are provided when the interpreter is invoked
- For example, the following invocation of the interpreter passes an array of three `String` objects into main method:

```
> java Calc 3 + 5
```

- The strings "3", "+", "5", are stored at indexes 0-2 of the String array `args`

Exercise: Implement a simple command-line calculator:

`<num1> +|-|*|- <num2>`

See Calc.java

Example: Command-Line Arguments

- Modify CaesarFile to take command, key, and file in commandline
  % java CaesarFileCL <enc|dec> <file> <key>

Outline

- Admin and recap
- Arrays
  - Motivation
  - Basic syntax
    - declare arrays, access array elements, OOB
    - array initializer list
    - array as method parameter/return
  - Using arrays
    - basic array loops to process array elements

create an array with random values	`double[] a = new double[N];`
	`for (int i = 0; i < N; i++)`
	`  a[i] = Math.random();`

print the array values, one per line	`for (int i = 0; i < N; i++)` `  System.out.println(a[i]);`

find the maximum of the array values	`double max = Double.NEGATIVE_INFINITY;`
	`for (int i = 0; i < N; i++)`
	`  if (a[i] > max) max = a[i];`

compute the average of the array values	`double sum = 0.0;`
	`for (int i = 0; i < N; i++)`
	`  sum += a[i];`
	`double average = sum / N;`

copy to another array	`double[] b = new double[N];`
	`for (int i = 0; i < N; i++)`
	`  b[i] = a[i];`

reverse the elements within an array	`for (int i = 0; i < N/2; i++)`
	`{`
	`  double temp = b[i];`
	`  b[i] = b[N-1-i];`
	`  b[N-i-1] = temp;`
	`}`

Array and Loops Examples

Outline

- Admin and recap
- Arrays
  - Motivation
  - Basic syntax
    - declare arrays, access array elements, OOB
    - array initializer list
    - array as method parameter/return
  - Using arrays
    - basic array loops
    - using array as counters/accumulators

Example Array Use Pattern: Array Elements as Counters/Accumulators

- Create an array equal to the size of the number of categories
- Loop over each input
  - map input's value to array index
  - increase the array element at index
- Display result
  - map each array index back to input to display

Grade Histogram Question

- Given a file of integer exam scores, where a score is between 0 and 100, e.g.,

```
82
66
79
63
83
```

Write a program that will print a histogram of stars indicating the number of students who earned each unique exam score.

```
85: *****
86: ***********
87: ***
88: *
91: ****
```

Example Array Use Pattern: Array Elements as Counters/Accumulators

- Create an array equal to the size of the number of categories
  - Q: how many categories?

```
int[] counters = new int[101];
```

- Loop over each input
  - map input's value to array index

```
grade is the index
```

  - increase the array element at index

- Display result
  - map each array index back to input to display

Grade Histogram

```
// Reads a file of test scores and shows a histogram of the score distribution.
import java.io.*;
import java.util.*;

public class GradeHistogram {
    public static void main(String[] args) throws FileNotFoundException {
        Scanner input = new Scanner(new File("midterm.txt"));
        int[] counts = new int[101];     // counters of test scores 0 - 100

        for (int i = 0; i < NSTUDENTS; i++) {     // read file into counts array
            int grade= input.nextInt();
            counts[grade]++;              // if gradeis 87, then counts[87]++
        }

        for (int i = 0; i < counts.length; i++) {    // print star histogram
            if (counts[i] > 0) {
                System.out.printf("%3d: ", i);
                for (int j = 0; j < counts[i]; j++) {
                    System.out.print("*");
                }
                System.out.println();
            }
        }
    }
}
```

Revision

- How about a bucket for every 5 points:
  - 00-04:
  - 05-09:
  - ..
  - 90-94:
  - 95-99:
  - 100:

Example Array Use Pattern: Array Elements as Counters/Accumulators

- Create an array equal to the size of the number of categories
  - Q: how many categories?

```
int[] counters = new int[100/5+1];
```

- Loop over each input
  - map input's value to array index

    grade -> index is grade/5
  - increase the array element at index

- Display result
  - map each array index back to input to display

    [index * 5, index*5+4] before 100

(Offline Exercise) Revision

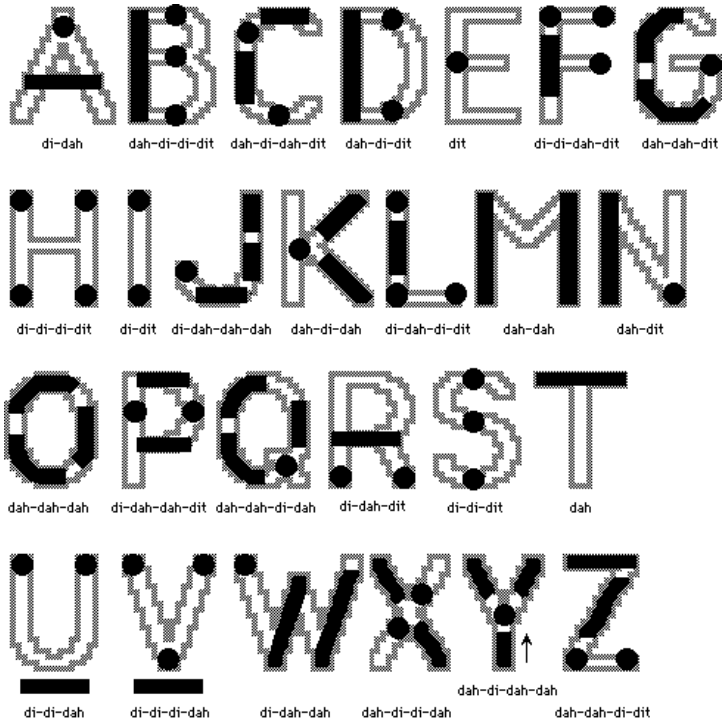
- How about the following buckets:
  - 00-59:
  - 60-64:
  - 65-69:
  - 70-74:
  - …
  - 90-94:
  - 95-99:
  - 100:

Letter Frequency Counting

- Objective: Count the frequency of letters a to z in a text file.

- The inventor of Morse code, Samuel Morse (1791-1872), counted letter frequencies to assign the simpler codes to the more frequently used letters. The counters he obtained:
  - E: 12000
  - T: 9000
  - A: 8000
  - …
  - X: 400
  - Z: 200

- Freq. counting is also the foundation of code breaking



Using Array as Counters/Accumulators

- Create an array equal to the size of the number of categories
  - Q: how many categories?

```
int[] counters = new int[26];
```

- Loop over each input
  - map input's value to array index

```
ch -> array index is ch-'a'
```

  - increase the array element at index

- Display result
  - map each array index back to input to display

```
index -> (char)('a'+ index)
```

Array Elements as Counters

- Count the number of characters in a line:

```
int[] counts = new int[26];
String line = scan.nextLine();

for (int i = 0; i < line.length(); i++) {

    char ch = line.charAt(i);

    if ('a' <= ch && ch <= 'z') {
        counts[ch-'a']++;
    }
}
```

Mapping data to index

Array Index to Display

```
public static void histogram(int[] counts) {

    for (int i = 0; i < counts.length; i++) {
        if (counts[i] > 0) {
            System.out.print( (char)('a' + i) + ": ");
            int h = counts[i];
            for (int j = 0; j < h; j++)
                System.out.print("*");
            }
            System.out.println();
        }
    }
} // end of histogram
```

Mapping index to data for better display

LetterHistogram.java

Outline

- Admin and recap
- Indefinite loops
  - Motivation
  - Program statements (for/while/do-while)

Motivation

- CaesarFile reads a file with a fixed number of lines. What if the file can contain an arbitrary number of lines of text?

Motivation

- CaesarFile reads a file with a fixed number of lines. What if the file can contain an arbitrary number of lines of text?
  - If there are more lines than the constant used in the loop, the remaining lines are not processed.
  - If there are fewer lines than the constant, program will crash

Outline

- Admin and recap
- Indefinite loops
  - Motivation
  - Program statements (for/while/do-while)

Indefinite Loops

- Number of times of a loop is not a simple number, but rather a logical condition
- **Indefinite loops** are common in programming design, e.g.,
  - Read from user until input is a non-negative number.
  - Repeat until the user has typed "q" to quit.
  - Print random numbers until a prime number is printed.
  - Search for a solution in a set.
- A key to define an indefinite loop is to identify the loop's test condition.

Scanner Test for Input Status

Method	Description
`hasNext()`	returns `true` if there is a next token
`hasNextInt()`	returns `true` if there is a next token and it can be read as an `int`
`hasNextDouble()`	returns `true` if there is a next token and it can be read as a `double`
`hasNextLine()`	returns `true` if there is a next line.

- These methods of the `Scanner` do not consume input; they just give information about what the next input will be.

CaesarFile using for Loop

```
public static void encode(Scanner scan, int key) {

   for( ; scan.hasNextLine(); )

        String text = scan.nextLine();

        String result = Caesar.encode(text, key);
        System.out.println( result );
   }
}
```

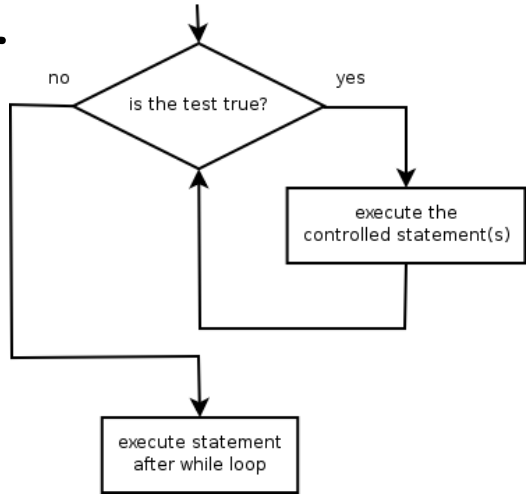
reads ugly

Java Repetition /Loop Statements

- Java has three kinds of loop statements:
  - the **`for`** *loop*
  - the **`while`** *loop*,
  - the **`do`** *loop*, and
- They have the same expressive power
- Which loop statements to use will depend on the context: pick the one that is more intuitive to read

The `while` loop

- **`while` loop**: Repeatedly executes its body as long as a logical test is true.



while is a keyword

```
while (<test>) {
    <statement(s)>;
}
```

- Example:

```
int num = 1;                          // initialization
while (num <= 10) {                   // test
    System.out.print(num + " ");
    num = num * 2;                    // update
}

// output:  1 2 4 8
```

The `do/while` loop

- **`do/while` loop**: Similar to while, except move test at the end:

```
do {
    <statement(s)>;
} while (<test>);
```

- Example:

```
int num = 1;                          // initialization
do {
    System.out.print(num + " ");
    num = num * 2;                    // update
} while (num <= 10);                  // test
// output:  1 2 4 8
```

CaesarFile using a while Loop

```
public static void encode(Scanner scan, int key) {

  while ( scan.hasNextLine() ) {

      String text = scan.nextLine();

      String result = Caesar.encode(text, key);
      System.out.println( result );
  }
}
```

Summary: CaesarFile using Loop

```
public static void encode(Scanner scan, int key) {

  for ( ; scan.hasNextLine(); )

      String text = scan.nextLine();
      String result = Caesar.encode(text, key);
      System.out.println( result );
  }}
```

reads ugly

```
public static void encode(Scanner scan, int key) {

  while ( scan.hasNextLine() ) {

      String text = scan.nextLine();
      String result = Caesar.encode(text, key);
      System.out.println( result );
  }
}
```

Summary: Flow Control Statements

- Loop statements
  - for
  - while
  - do/while
- Choosing the loop control structure depends on the specific situation and personal taste

```
for ( initialization ;
      condition ;
      increment ) {
   statement list;
}
```

```
while ( condition ) {
   statement list;
}
```

```
do {
   statement list;
} while ( condition );
```

Summary: `for` loop

- typically used in sequential iteration
- `for` loop is easy to read, since it provides visual aid to recognize the four components
  - If complex or empty initialization, increment, or condition, a `for` loop may not read as well as a `while` or `do/while` loop

```
for ( initialization ;
      condition ;
      increment ) {
   statement list;
}
```

Summary: while, do/while loop

- The choice between do/while and while depends on the logic of the program:
  - first check condition or first execute statement

```
while ( condition ) {
    statement list;
}
```

```
do
{
    statement list;
} while ( condition );
```

(Offline Exercise): Coupon Collector

- Write a program to simulate the number of coupons to collect before collecting at least one of each N distinct coupons

CouponCollector.java

Roadmap: Program Flow Control

- We have covered all syntax of program flow control
- Mastering program control structure is among the most important in terms of fully mastering computer programming
  - Program control structure may involve complex logical analysis
  - Program control structure often has substantial flexibility and hence depends on personal style
    - multiple types of conditional statements
      - if/else; switch; conditional operator
    - multiple types of loop statements
      - for, while, do/while
    - break/return in the middle of a loop

How to Grasp Program Flow Control

- Learn common loop patterns
- Conduct program analysis
- Read and practice
  - Try out the offline exercises
  - Read sample programs

Outline

- Admin and recap
- Indefinite loops
  - Motivation
  - Program statements (for/while/do/while)
  - Common indefinite loop pattern: search loop

Example

- Design a method `isInt`: returns `true` if all chars of a string are numbers, e.g.,
  - `isInt("4822116")` returns `true`
  - `isInt("2016Mar3")` returns `false`

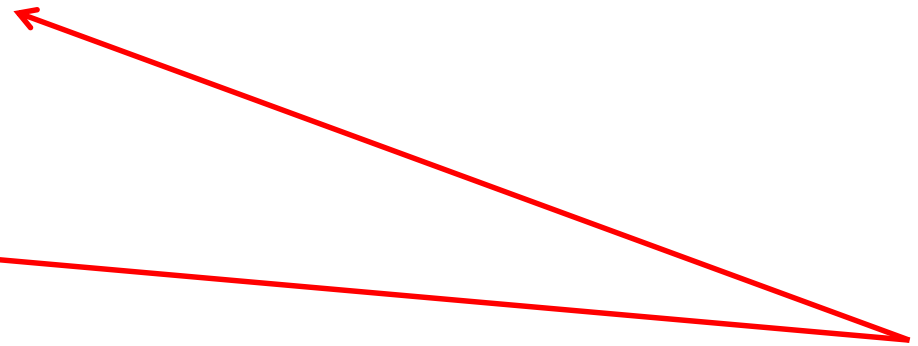
- Design questions:
  - for, while, or do while loop?
  - At each loop iteration (over a char), what conclusion can we have?
    - if the char is not a number
      - false; An evidence that is enough to draw conclusion; need to break (return)
    - if the char is a number
      - not enough evidence, need to continue

Boolean Return Answer

```
public static boolean isInt(String word) {
   for (int i = 0; i < word.length(); i++) {
      if (! isDigit( word.charAt(i) ) {
         return false;
      }
   } // end of for
   return true;
}

public static boolean isDigit(char ch) {
    return '0' <= ch && ch <= '9';
}
```

Alternative Design

```
public static boolean isInt(String word) {
   for (int i = 0; i < word.length(); i++) {
      if (! isDigit( word.charAt(i) ) {
         return false;
      }
   } // end of for
   return true;
}
```



Multiple exit points

Some programmers do not like a method having multiple exit (return) points. Objective: design a single exit point.

Analysis

```
public static boolean isInt(String word) {
   for (int i = 0; i < word.length(); i++) {
      if (! isDigit( word.charAt(i) ) {
         return false;
      }
   } // end of for
   return true;
}
```

- Question: logical conditions that the `for` loop should exit?

```
i >= word.length  || !isDigit(i)
```

Condition to continue the loop:

```
!( i >= word.length  || ! isDigit(i) )
```

De Morgan's Law

- **De Morgan's Law**: Rules used to negate boolean tests.
  - Useful when you want the opposite of an existing test.

Original Expression	Negate	Equivalent Negate
`a && b`	`!(a && b)`	`!a || !b`
`a || b`	`!(a || b)`	`!a && !b`

```
!( i >= word.length  || ! isDigit(i) )
```

```
( i < word.length )  &&  isDigit(i)
```

Alternative Version

```
public static boolean isInt(String word) {
   for (int i = 0; i < word.length(); i++) {
      if (! isDigit( word.charAt(i) ) {
         return false;
      }
   } // end of for
   return true;
}
```

```
public static boolean isInt(String word) {
  for (int i = 0;
       i < word.length() && isDigit(word.charAt(i));
      i++) { }
  // ??? return;
}
```

Alternative Version (any issues?)

```
public static boolean isInt(String word) {
  for (int i = 0;
       i < word.length() && isDigit(word.charAt(i));
       i++) { }
  // ??? return;
}
```

```
public static boolean isInt(String word) {
   for (int i = 0;
        i < word.length() && isDigit(word.charAt(i));
       i++) { }
   if (i < word.length())
      return false;
   else return true;
}
```

Alternative Version

```
public static boolean isInt(String word) {
  int i = 0;
  for (;
      i < word.length() && isDigit(word.charAt(i));
      i++) { }
  return i >= word.length();
}
```

Alternative Version

```
public static boolean isInt(String word) {
  int i = 0;
  for (;
      i < word.length() && isDigit(word.charAt(i);
      i++) { }
  return i >= word.length();
}
```

```
public static boolean isInt(String word) {
 int i = 0;
 while (i < word.length() && isDigit(word.charAt(i)) {
     i++;
 }
 return i >= word.length() ;
}
```

(Offline) Exercise

- Design a method `hasAnOddDigit` : **returns** `true` if any digit of an integer n is odd, e.g.,
  - `hasAnOddDigit(4822116)` **returns** `true`
  - `hasAnOddDigit(2448)` **returns** `false`

- Design questions:
  - How do we loop over each digit from n

```
digit = n % 10;
n = n / 10;
```

  - How do we stop
    - If the digit we just saw is odd, can we draw any conclusion?

      `Yes. We found an evidence, return true`

    - If the digit is even, can we draw any conclusion?

      `No, unless we have seen all digits.`

    - When do we know we have seen all digits?

Boolean Return Answer

```
public static boolean hasAnOddDigit(int n) {
    while (n != 0) {
        int digit = n % 10;
        if (digit % 2 != 0) { // find an example,
                              // enough to draw conclusion
            return true;
        }
        n = n / 10;
    }
    return false;           // if we reach here, no
                            // evidence of odd, return false
}
```

```
public static boolean hasAnOddDigit(int n) {
    while (n != 0) {
        if (n % 2 != 0) { // find an example,
                          // enough to draw conclusion
            return true;
        }
        n = n / 10;
    }
    return false;           // if we reach here, no
                            // evidence of odd, return false
}
```

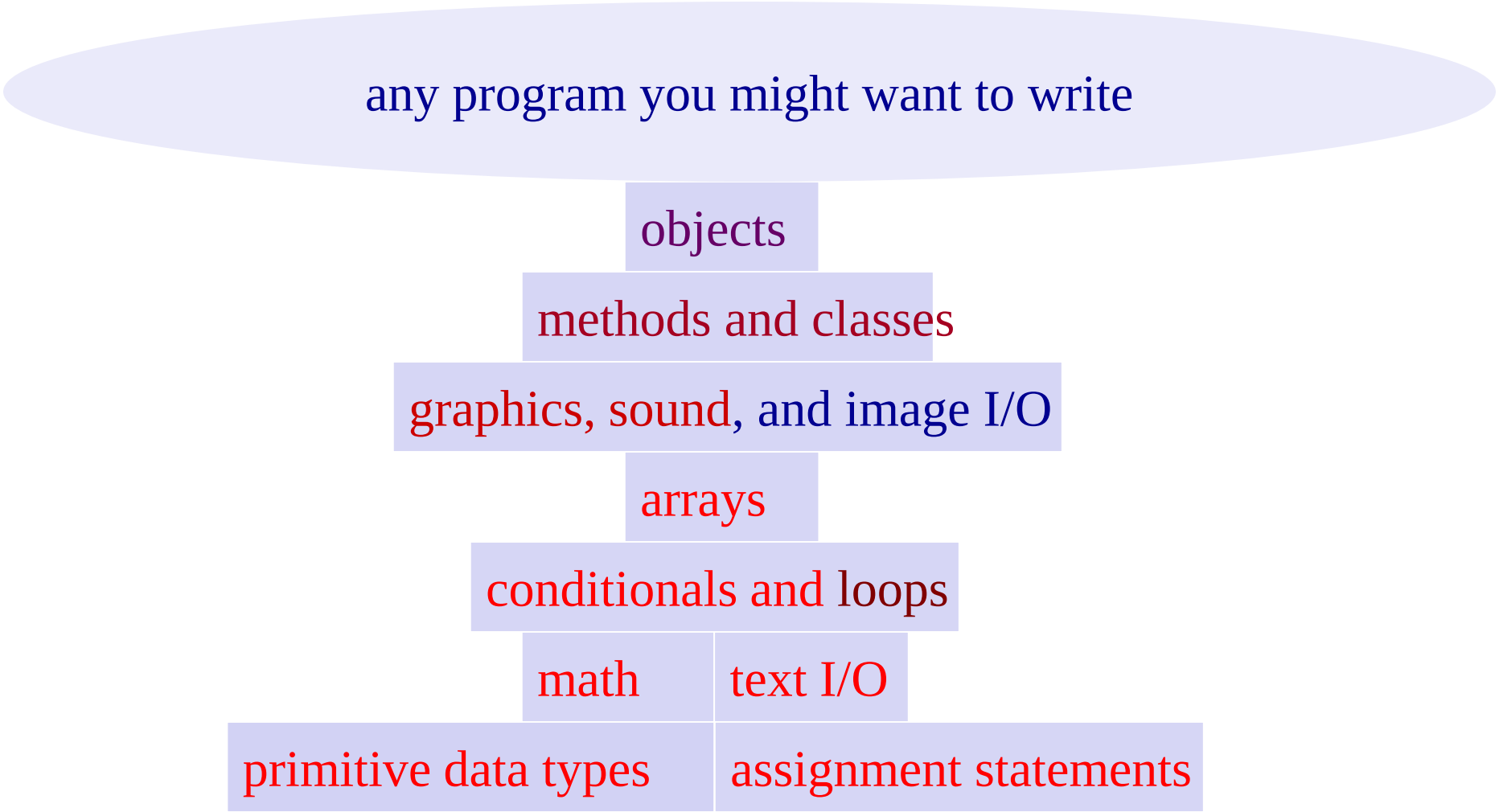
(Offline) Exercise

- Designs method `allDigitsOdd` : returns `true` if every digit of an integer is odd.
  - `allDigitsOdd(135319)` **returns** `true`
  - `allDigitsOdd(9174529)` **returns** `false`

Boolean Return Answer

```
public static boolean allDigitsOdd(int n) {
    while (n != 0) {
        if (n % 2 == 0) { // find a counter example,
                          // enough to draw conclusion
            return false;
        }
        n = n / 10;
    }

    return true;          // if we reach here, no
                          // evidence of counter example,
                          // return true
}
```

Foundational Programming Concepts

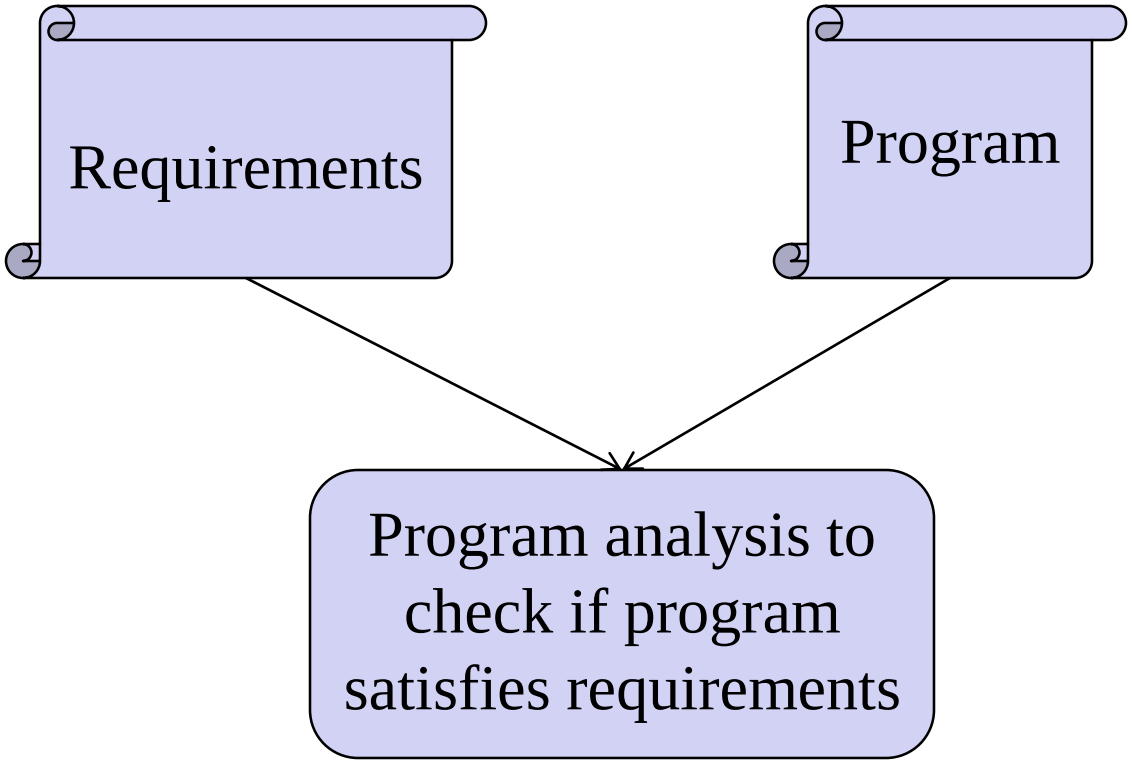


Outline

- Admin and recap
- Indefinite loops
  - Motivation
  - Program statements (for/while/do-while)
  - Common indefinite loop pattern: search loop
  - Program analysis

Program Analysis

- A useful tool to understand flow control



Foundation of Program Analysis: Logical Assertions

- **Assertion**: A statement that we focus on whether it is true, false, or sometime true/sometime false (unknown).

  Examples:

  - Yale was founded in 1701.
  - The capital of Connecticut is New Haven.
  - Prof. Yang met a donkey this morning.
  - int x;

    ...

    x = x+10;

    x divided by 2 equals 7. *(depends on the value of x)*

Logical Assertions on Program Variables

- One can make assertions on program variables at each point of a program
  - For example, right after a variable is initialized, its value is known, and we can make true/false statement:

```
int x = 3;
// is x > 0?  True
```

- A common confusion: An assertion is not part of your program; it is an external claim/property of your program

Difficulty of Making Assertions

- The value of a variable may become unknown after certain operations (hence leading to "unknown/sometimes" assertions), e.g.,
  - reading from a `Scanner`
  - assigned a number from a random number
  - a parameter's initial value to a method

```
public static void mystery(int a, int b)
{
// is a == 10?  UNKNOWN
```

Control Structure Establishes Assertions

- A key function of a control structure (e.g., `if`, `while`, `break`) is that it establishes assertions:

```
public static void mystery(int a, int b)
{
    if (a < 0) {
        // is a == 10?    FALSE
        ...
    }
}
```

We know a < 0, which implies a != any positive number.

Assertions and Controls

- At the start of an `if` or loop's body, the <test> or what can be implied by the <test> must be `true`, e.g.,

```
while (y < 10) {
    // is y < 10?   TRUE
    ...
}
```

- In the `else` or after a loop w/o break, the <test> must be `false`:

```
while (y < 10) {
    ...
}
// is y < 10?        FALSE
```

- Note: Inside a loop's body, the loop's test may become `false`:

```
while (y < 10) {
    y++;
    // is y < 10?   UNKNOWN
    // if y < 12     TRUE
}
```

Using Assertions to Understand Program

```
public static double getNonNegDouble(Scanner console) {
    System.out.print("Type a nonnegative number: ");
    double number = console.nextDouble();

    while (number < 0.0) {
       // ASSERTION: number < 0
       System.out.print("Error: Negative; try again: ");

       number = console.nextDouble();
    }
    // ASSERTION: number >= 0.0
    return number;
}
```

Is the following statement about the program above correct:

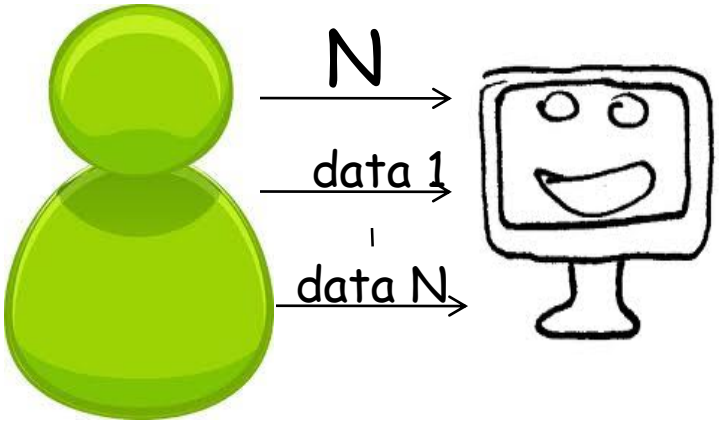
- it prints an error message only if user `number` is negative
- it returns `number` only if non-negative

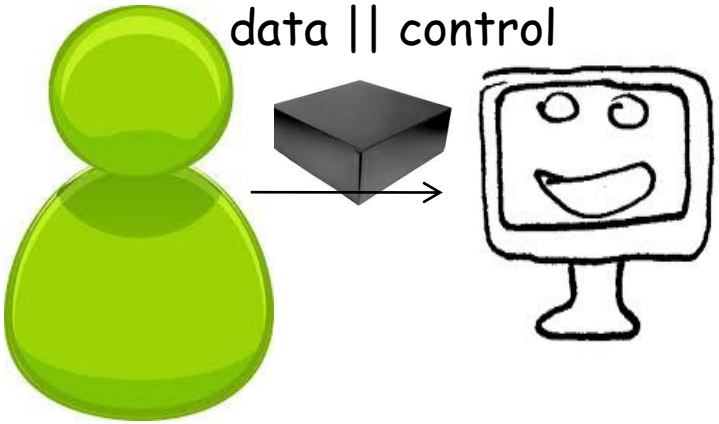
Outline

- Admin and recap
- Indefinite loops
  - Motivation
  - Program statements (for/while/do/while)
  - Common indefinite loop pattern: search loop
  - Program analysis
  - Common indefinite loop pattern: sentinel user input processing

User Input Protocol

- Two input styles
  - Header control protocol
    - User first specifies the number of data items (e.g., TaskMan)
  - In-band control protocol
    - User finishes input by entering a sentinel value
      - e.g., -1 to signal the end of input grades; "quit" to finish the program
    - Why in-band sentinel: flexibility.
    - Complexity of in-band sentinel: a data item just read can be either a real data item or the signaling sentinel





Sentinel Values

- **sentinel**: A value that signals the end of user input.
  - **sentinel loop**: Repeats until a sentinel value is seen.

- Example: Write a program that prompts the user for exam grade until the user types -1, then outputs the average grade.
  - (In this case, the value -1 is the sentinel value.)

```
Type a grade (-1 to exit): 100
Type a grade (-1 to exit): 90
Type a grade (-1 to exit): -1
The average is 95.
```

Design question: for, while, or do loop?

Potential Solution

```
Scanner console = new Scanner(System.in);
int sum = 0, count = 0;
int grade;

do {
  System.out.print("Type a grade (-1 to exit): ");
  grade = console.nextInt();

  sum += grade; count ++;
} while (grade != -1);

System.out.println(”The average is “ +
                   (count > 0? sum /count : 0) );
```

Potential Solution: Test

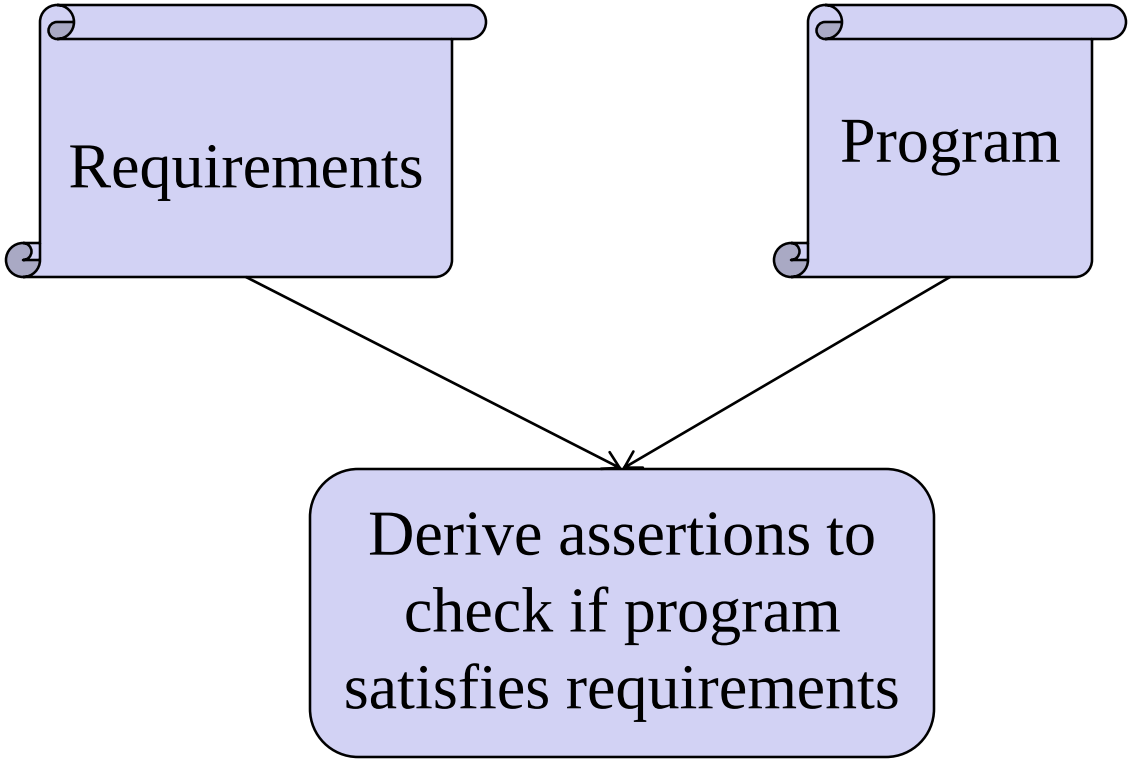
```
Scanner console = new Scanner(System.in);
int sum = 0, count = 0;
int grade;
do {
  System.out.print("Type a grade (-1 to exit): ");
  grade = console.nextInt();

  sum += grade; count ++;
} while (grade != -1);

System.out.println("The average is " +
                   (count > 0? sum /count : 0) );
```

```
Type a grade (-1 to exit): 100
Type a grade (-1 to exit): 90
Type a grade (-1 to exit): -1
```

Output: `The average is 63`

The output is incorrect!

Program Analysis



Requirements Specification

The program reads in a sequence of grades, adds each non-sentinel grade to sum/count, stops when sees sentinel.

reads in a non-sentinel `grade` => add it to sum/count

reads in sentinel `grade` => not add to sum/count; read loop must stop

Equivalent cond for safety:

add to sum/count => grade is not sentinel

Program Analysis

```
Scanner console = new Scanner(System.in);
int sum = 0, count = 0;
int grade;
do {
  System.out.print("Type a grade (-1 to exit): ");
  grade = console.nextInt();
  // assertion grade != -1 must be established
  sum += grade; count ++;
} while (grade != -1);

System.out.println("The average is " +
                   (count > 0? sum /count : 0) );
```

Condition to guarantee safety:

add to sum and count => grade != -1

Program Revision to Establish Assertion

```
Scanner console = new Scanner(System.in);
int sum = 0, count = 0;
int grade;
do {
  System.out.print("Type a grade (-1 to exit): ");
  grade = console.nextInt();
  if (grade != -1) {
  sum += grade; count ++;
  }
} while (grade != -1);

System.out.println("The average is " +
                   (count > 0? sum /count : 0) );
```

Version 2: do/while -> while

```
Scanner console = new Scanner(System.in);
int sum = 0, count = 0;
int grade;
while (grade != -1) {
  System.out.print("Type a grade (-1 to exit): ");
  grade = console.nextInt();
  if (grade != -1) {
  sum += grade; count ++;
  }
}

System.out.println("The average is " +
                   (count > 0? sum /count : 0) );
```

Final Version 2: do/while -> while

```
Scanner console = new Scanner(System.in);
int sum = 0, count = 0;
int grade = 0; // any value other than sentinel
while (grade != -1) {
  System.out.print("Type a grade (-1 to exit): ");
  grade = console.nextInt();
  if (grade != -1) {
  sum += grade; count ++;
  }
}

System.out.println("The average is " +
                   (count > 0? sum /count : 0) );
```

Outline

- Admin and recap
- Indefinite loops
  - Motivation
  - Program statements (for/while/do/while)
  - Common indefinite loop pattern: search loop
  - Program analysis
  - Common indefinite loop pattern: sentinel user input processing
    - Basic design using assertion to view and fix bug
    - An alternative view of the bug: a design pattern

A "Simpler" Problem...

- Revisit the `countDown` method that prints from a given maximum (>=1) to 1, separated by commas.

  For example, the call:

  ```
  countDown(5)
  ```

  should print:

  ```
  5, 4, 3, 2, 1
  ```

Previous "Solution"

5, 4, 3, 2, 1

```
public static void countDown(int max) {
    for (int i = max; i >= 1; i--) {
        System.out.print(i + ", ");
    }
    System.out.println();  // to end the line of output
}
```

- Output from `countDown(5)`: 5, 4, 3, 2, 1,

Previous "Solution"

5, 4, 3, 2, 1

```
public static void countDown(int max) {
    for (int i = max; i >= 1; i--) {
        System.out.print(i + ", ");
    }
    System.out.println();  // to end the line of output
}
```

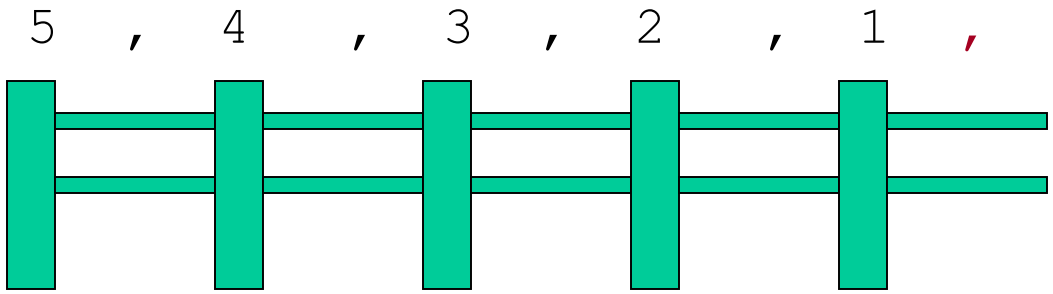
- Output from `countDown(5)`: 5, 4, 3, 2, 1,

```
public static void countDown(int max) {
    for (int i = max; i >= 1; i--) {
        System.out.print(", " + i);
    }
    System.out.println();  // to end the line of output
}
```

- Output from `countDown(5)`: , 5, 4, 3, 2, 1

Problem: Fence Post Analogy

- We print *n* numbers but need only *n* - 1 commas.

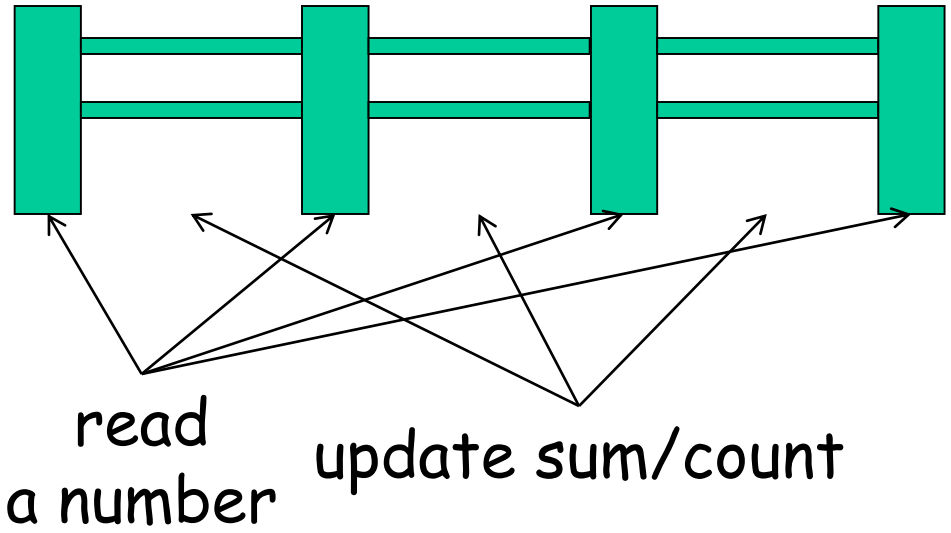


- If we use a loop algorithm that repeatedly places a post + wire, the last post will have an extra dangling wire.

```
loop (length of fence-wire pair) {
    place a post.        // e.g., <number>
    place some wire.     // e.g., ,
}
```

Problem: Fence Post Analogy

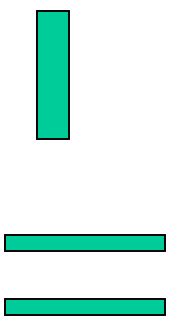
The sentinel input loop pattern is also a fencepost problem: Must read *N* grades, but sum/count only the first *N*-1.



Problem: Fence Post Analogy

The sentinel input loop pattern is also a fencepost problem: Must read *N* grades, but sum/count only the first *N*-1.

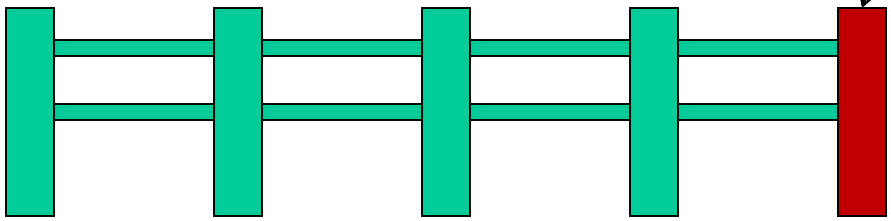
```
Scanner console = new Scanner(System.in);
int sum = 0, count = 0;
int grade;
do {
  System.out.print("Type a grade (-1 to exit): ");
  grade = console.nextInt();


  sum += grade; count ++;

} while (grade != -1);

System.out.println("The average is " +
                   count > 0? sum /count : 0);
```



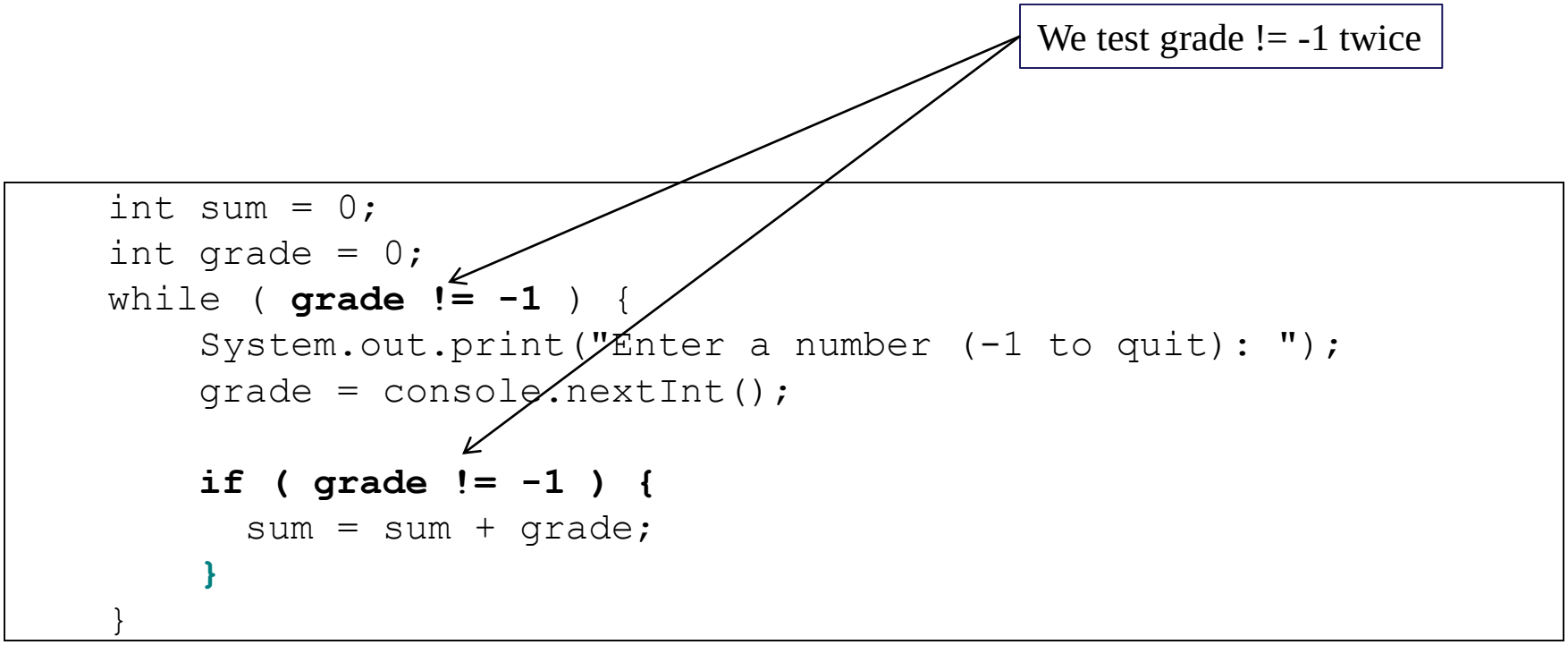
Solve the Fencepost Problem: Design I

```
loop (length of fence) {
    place a post.
    if (! last post)
        place a wire.
}
```

Detect if it is the last post in the loop, if it is, do not place the wire



Revisit Previous Program




```
int sum = 0;
int grade = 0;
while ( grade != -1 ) {
    System.out.print("Enter a number (-1 to quit): ");
    grade = console.nextInt();

    if ( grade != -1 ) {
      sum = sum + grade;
    }
}
```

Q: What does the if() do?

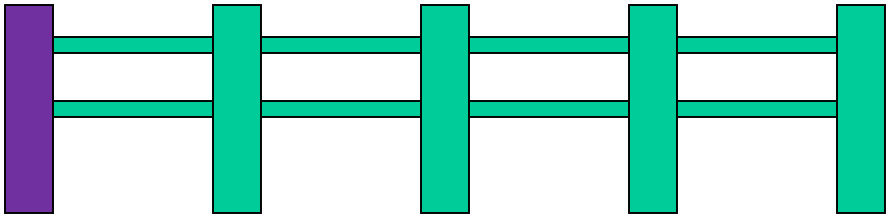
Alternative: Sentinel Loop with `break`

```
Scanner console = new Scanner(System.in);
int sum = 0, count = 0;
int grade;
do {
  System.out.print("Type a grade (-1 to exit): ");
  grade = console.nextInt();
  if (grade == -1)
    break; // break stops the containing loop
  // Do we have assertion grade != -1?
  sum += grade; count ++;

} while (grade != -1);

System.out.println("The average is " +
                   (count > 0? sum /count : 0) );
```

Design Pattern II

- Add a statement outside the loop to place the initial "post." Also called a "loop-and-a-half" solution.

```
place first post.
loop (length of fence - 1) {
    place some wire.
    place a post.
}
```



Fencepost Method Solution

```
public static void countDown(int max) {
    System.out.print(max);    // first post
    for (int i = max-1; i >= 1; i--) {
        System.out.print(", " + i); // wire + post
    }
    System.out.println();     // to end the line
}
```

- Alternate solution: Either first or last "post" can be taken out:

```
public static void countDown(int max) {
    for (int i = max; i >= 2; i--) {
        System.out.print(i + ", "); // post + wire
    }
    System.out.println(1);  // last post
}
```

Fencepost Sentinel Loop: Grade

```
public static final int SENTINEL = -1;
public static final String PROMPT = "Type a grade ("
                                    + SENTINEL + " to exit): ";
public static GradeAnalyzer() {
  Scanner console = new Scanner(System.in);
  int sum = 0; int count = 0;

  // pull one prompt/read ("post") out of the loop
  int grade = getInt(PROMPT, console);

  while ( grade != SENTINEL ) {
    // Do we have assertion grade != -1 before updates?
    sum  += grade; count++;                    // wire
    grade = getInt(PROMPT, console);           // post
  }

  if (count > 0)
     System.out.println("Avg: "  1.0 * sum / count);
}

public static String getInt(String prompt, Scanner console)
{   System.out.print(prompt);
    return console.nextInt();
}
```

Comparison

```
int sum = 0;
System.out.print("Enter a number (-1 to quit): ");
int grade = console.nextInt();

while ( grade != -1 ) {
    sum = sum + grade;

    System.out.print("Enter a number (-1 to quit): ");
    grade = console.nextInt();
}
```

Duplicated code

```
int sum = 0;
int grade = 0;
while ( grade != -1 ) {
    System.out.print("Enter a number (-1 to quit): ");
    grade = console.nextInt();

    if ( grade != -1 ) {      // detect the last post
        sum = sum + grade;
    }
}
```

Duplicated code
Duplicate execution

Offline Exercise

- Design loops without duplicates