ITP20003 Java Programming

Basic Operations (Chapter 2)

This slide is primary taken from the instructor's resource of Java: Introduction to Problem Solving and Programming, 7th ed. by Savitch and then edited partly by Shin Hong

Variables

- *Variables* store data such as numbers and letters.
  - Think of them as places to store data.
  - They are implemented as memory locations.
- The data stored by a variable is called its *value.*
  - The value is stored in the memory location.
- Its value can be changed.

Variables

- View sample program listing 2.1
  - **`Class EggBasket`**

```
If you have
6 eggs per basket and
10 baskets, then
the total number of eggs is 60
```

Sample Screen Output

Example

```
public class EggBasket
{
    public static void main (String [] args)
    {
      int numberOfBaskets, eggsPerBasket, totalEggs;
      numberOfBaskets = 10;
      eggsPerBasket = 6;
      totalEggs = numberOfBaskets * eggsPerBasket;
      System.out.println ("If you have");
      System.out.println (eggsPerBasket + " eggs per basket and");
      System.out.println (numberOfBaskets + " baskets, then");
      System.out.println ("the total number of eggs is " +
                          totalEggs);
    }
}
```

Variables and Values

- Variables

```
numberOfBaskets
eggsPerBasket
totalEggs
```

- Assigning values

```
eggsPerBasket = 6;
eggsPerBasket = eggsPerBasket - 2;
```

Naming and Declaring Variables

- Choose names that are helpful such as `count` or `speed`, but not `c` or `s`.
- When you *declare* a variable, you provide its name and type.

```
int numberOfBaskets,eggsPerBasket;
```

- A variable's *type* determines what kinds of values it can hold (`int`, `double`, `char`, etc.).
- A variable must be declared before it is used.

Syntax and Examples

- Syntax

  `type variable_1, variable_2, …;`

  (`variable_1` is a generic variable called a *syntactic variable*)

- Examples

```
int styleChoice, numberOfChecks;
double balance, interestRate;
char jointOrIndividual;
```

Data Types

- A *class type* is used for a class of objects and has both data and methods.
  - `"Java is fun"` is a value of class type `String`

- A *primitive type* is used for simple, non-decomposable values such as an individual number or individual character.
  - `int`, `double`, and `char` are primitive types.

Primitive Types

FIGURE 2.1 Primitive Type

Type Name	Kind of Value	Memory Used	Range of Values
`byte`	Integer	1 byte	–128 to 127
`short`	Integer	2 bytes	–32,768 to 32,767
`int`	Integer	4 bytes	–2,147,483,648 to 2,147,483,647
`long`	Integer	8 bytes	–9,223,372,036,8547,75,808 to 9,223,372,036,854,775,807
`float`	Floating-point	4 bytes	$\pm 3.40282347 \times 10^{+38}$ to $\pm 1.40239846 \times 10^{-45}$
`double`	Floating-point	8 bytes	$\pm 1.79769313486231570 \times 10^{+308}$ to $\pm 4.94065645841246544 \times 10^{-324}$
`char`	Single character (Unicode)	2 bytes	All Unicode values from 0 to 65,535
`boolean`		1 bit	True or false

Java Identifiers

- An *identifier* is a name, such as the name of a variable
- Identifiers may contain only
  - Letters
  - Digits (0 through 9)
  - The underscore character (_)
  - And the dollar sign symbol ($) which has a special meaning
- The first character cannot be a digit.

Java Identifiers

- Identifiers may not contain any spaces, dots (.), asterisks (*), or other characters:

  `7-11` `oracle.com` `util.*` (not allowed)

- Identifiers can be arbitrarily long.
- Since Java is *case sensitive*, `stuff`, `Stuff`, and `STUFF` are different identifiers.

Keywords or Reserved Words

- Words such as `if` are called *keywords* or *reserved words* and have special, predefined meanings.
  - Cannot be used as identifiers.
  - See Appendix 1 for a complete list of Java keywords.
- Example keywords: `int`, `public`, `class`

Naming Conventions

- Class types begin with an uppercase letter (e.g. `String`).
- Primitive types begin with a lowercase letter (e.g. `int`).
- Variables of both class and primitive types begin with a lowercase letters (e.g. `myName`, `myBalance`)
- Multiword names are "punctuated" using uppercase letters.

Where to Declare Variables

- Declare a variable
  - Just before it is used or
  - At the beginning of the section of your program that is enclosed in `{}`.

```
public static void main(String[] args)
{ /* declare variables here */
    . . .
}
```


JAVA: An Introduction to Problem Solving & Programming, 7th Ed. By Walter Savitch
ISBN 0133862119

Primitive Types

- Four integer types (`byte`, `short`, `int`, and `long`)
  - `int` is most common
- Two floating-point types (`float` and `double`)
  - `double` is more common
- One character type (`char`)
- One boolean type (`boolean`)

Examples of Primitive Values

- Integer types

```
0   -1   365   12000
```

- Floating-point types

```
0.99   -22.8   3.14159 5.0
```

- Character type

```
'a'   'A'   '#'   ' '
```

- Boolean type

```
true   false
```

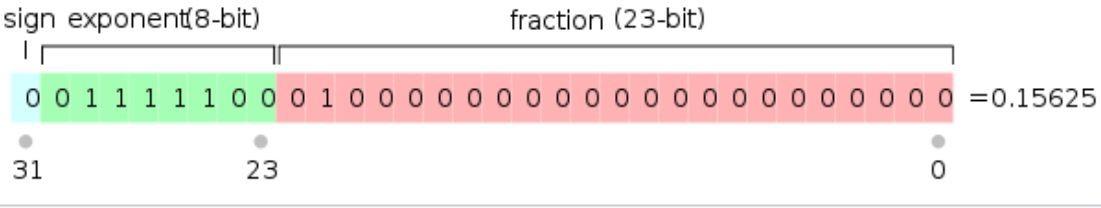
e Notation

- e notation is also called *scientific notation* or *floating-point notation.*
- Examples
  - `865000000.0` can be written as `8.65e8f` or `8.65e8d`
  - `0.000483` can be written as `4.83e-4f` or `4.83e-4d`
- The number in front of the `e` does not need to contain a decimal point.

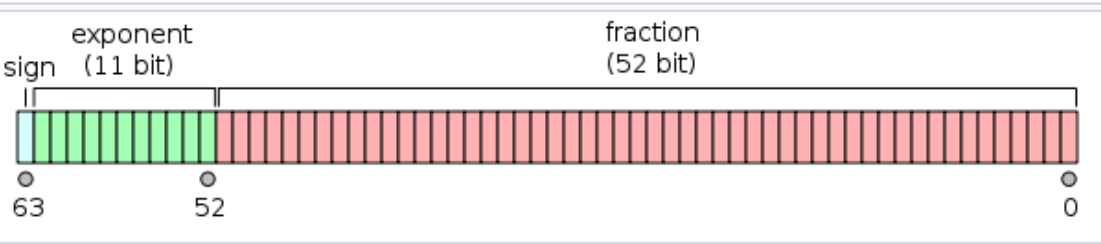
Floating Number Representation

- Ref. https://en.wikipedia.org/wiki/IEEE_754-1985

$$(-1)^{b_{31}} \times 2^{(b_{30}b_{29}\ldots b_{23})_2-127} \times (1.b_{22}b_{21}\ldots b_0)_2$$



The number 0.15625 represented as a single-precision IEEE 754-1985 floating-point number. See text for explanation.



The three fields in a 64bit IEEE 754 float

Imprecision in Floating-Point Numbers

- Floating-point numbers often are only approximations since they are stored with a finite number of bits.
- Hence `1.0/3.0` is slightly less than 1/3.
- `1.0/3.0 + 1.0/3.0 + 1.0/3.0` is less than 1.

Assignment Statements

- An assignment statement is used to assign a value to a variable.

```
answer = 42;
```

- The "equal sign" is called the *assignment operator.*
- We say, "The variable named **`answer`** is assigned a value of 42," or more simply, "**`answer`** is assigned 42."

Assignment Statements

- Syntax

  `variable = expression`

  where **`expression`** can be another variable, a *literal* or *constant* (such as a number), or something more complicated which combines variables and literals using *operators*
  (such as + and -)

Assignment Examples

```
amount = 3.99;
firstInitial = 'W';
score = numberOfCards + handicap;
eggsPerBasket = eggsPerBasket - 2;
```

Initializing Variables

- A variable that has been declared, but no yet given a value is said to be *uninitialized.*
- Java does not allow a variable remained as uninitialized before its possible first use
- Java enforces a developer to assign a value at the time the variable is declared

Initializing Variables

- Syntax

```
type variable_1 = expression_1,
variable_2 = expression_2, …;
```

- Examples:

```
int count = 0;
char grade = 'A';
```


JAVA: An Introduction to Problem Solving & Programming, 7th Ed. By Walter Savitch
ISBN 0133862119

Assignment Evaluation

- The expression on the right-hand side of the assignment operator (=) is evaluated first.
- The result is used to set the value of the variable on the left-hand side of the assignment operator.

```
score = numberOfCards +  handicap;
eggsPerBasket = eggsPerBasket - 2;
```

Simple Input

- Sometimes the data needed for a computation are obtained from the user at run time.
- Keyboard input requires

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

at the beginning of the file.

Simple Input

- Data can be entered from the keyboard using

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

  followed, for example, by

```
eggsPerBasket =
keyboard.nextInt();
```

  which reads one `int` value from the keyboard and assigns it to `eggsPerBasket`.

```java
import java.util.Scanner;
public class EggBasket2 {
    public static void main (String [] args)     {
        int numberOfBaskets, eggsPerBasket, totalEggs;
        Scanner keyboard = new Scanner (System.in);
        System.out.println ("Enter the number of eggs in each basket:");
        eggsPerBasket = keyboard.nextInt ();
        System.out.println ("Enter the number of baskets:");
        numberOfBaskets = keyboard.nextInt ();
        totalEggs = numberOfBaskets * eggsPerBasket;
        System.out.println ("If you have");
        System.out.println (eggsPerBasket + " eggs per basket and");
        System.out.println (numberOfBaskets + " baskets, then");
        System.out.println ("the total number of eggs is " + totalEggs);
        System.out.println ("Now we take two eggs out of each basket.");
        eggsPerBasket = eggsPerBasket - 2;
        totalEggs = numberOfBaskets * eggsPerBasket;
        System.out.println ("You now have");
        System.out.println (eggsPerBasket + " eggs per basket and");
        System.out.println (numberOfBaskets + " baskets.");
        System.out.println ("The new total number of eggs is " + totalEggs);
    }
}
```

Simple Screen Output

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

- Outputs the sting literal `"the count is "`
- Followed by the current value of the variable `count`.

Constants

- Literal expressions such as `2`, `3.7`, or `'y'` are called *constants*.
- Integer constants can be preceded by a `+` or `–` sign, but cannot contain commas.
- Floating-point constants can be written
  - With digits after a decimal point or
  - Using *e notation*

Imprecision in Floating-Point Numbers

- Floating-point numbers often are only approximations since they are stored with a finite number of bits.
- Hence `1.0/3.0` is slightly less than 1/3.
- `1.0/3.0 + 1.0/3.0 + 1.0/3.0` is less than 1.

Named Constants

- Java provides mechanism to …
  - Define a variable
  - Initialize it
  - Fix the value so it cannot be changed

```
public static final Type Variable = Constant;
```

  - Example

```
public static final double PI = 3.14159;
```

Assignment Compatibilities

- Java is said to be *strongly typed.*
  - You can't, for example, assign a floating point value to a variable declared to store an integer.
- Sometimes conversions between numbers are possible.

```
doubleVariable = 7;
```

is possible even if `doubleVariable` is of type `double`, for example.

Assignment Compatibilities

- A value of one type can be assigned to a variable of any type further to the right

```
byte --> short --> int --> long
--> float --> double
```

  - But not to a variable of any type further to the left.
- You can assign a value of type char to a variable of type `int`.

Type Casting

- A *type cast* temporarily changes the value of a variable from the declared type to some other type.
- For example,

```
double distance;
distance = 9.0;
int points;
points = (int)distance;
```

- Illegal without `(int)`

JAVA: An Introduction to Problem Solving & Programming, 7th Ed. By Walter Savitch
ISBN 0133862119

Type Casting

- The value of `(int)distance` is `9`,
- The value of `distance`, both before and after the cast, is `9.0`.
- Any nonzero value to the right of the decimal point is *truncated* rather than *rounded.*

Arithmetic Operators

- Arithmetic expressions can be formed using the `+`, `-`, `*`, and `/` operators together with variables or numbers referred to as *operands.*
  - When both operands are of the same type, the result is of that type.
  - When one of the operands is a floating-point type and the other is an integer, the result is a floating point type.

Arithmetic Operations

- Example

  If **`hoursWorked`** is an **`int`** to which the value **`40`** has been assigned, and **`payRate`** is a **`double`** to which **`8.25`** has been assigned

```
hoursWorked * payRate
```

  is a **`double`** with a value of **`500.0`**.

Arithmetic Operations

- Expressions with two or more operators can be viewed as a series of steps, each involving only two operands.
  - The result of one step produces one of the operands to be used in the next step.
- example

```
balance + (balance * rate)
```

Arithmetic Operations

- If at least one of the operands is a floating-point type and the rest are integers, the result will be a floating point type.
- The result is the rightmost type from the following list that occurs in the expression.

```
byte --> short --> int --> long
--> float --> double
```

The Division Operator

- The division operator (`/`) behaves as expected if one of the operands is a floating-point type.
- When both operands are integer types, the result is truncated, not rounded.
  - Hence, `99/100` has a value of `0`.

The `mod` Operator

- The `mod` (`%`) operator is used with operators of integer type to obtain the remainder after integer division.
- 14 divided by 4 is 3 *with a remainder of 2.*
  - Hence, `14 % 4` is equal to `2`.
- The mod operator has many uses, including
  - determining if an integer is odd or even
  - determining if one integer is evenly divisible by another integer.

Parentheses and Precedence

- Parentheses can communicate the order in which arithmetic operations are performed
- examples:

```
(cost + tax) * discount
(cost + (tax * discount)
```

- Without parentheses, an expressions is evaluated according to the *rules of precedence.*

Precedence Rules

- Figure 2.2 Precedence Rules

Highest Precedence

First: the unary operators +, –, !, ++, and – –

Second: the binary arithmetic operators *, /, and %

Third: the binary arithmetic operators + and –

Lowest Precedence

Precedence Rules

- The *binary* arithmetic operators `*`, `/`, and `%`, have *lower precedence* than the *unary* operators `+`, `-`, `++`, `--`, and `!`, but have *higher precedence* than the binary arithmetic operators `+` and `-`.
- When binary operators have equal precedence, the operator on the left acts before the operator(s) on the right.

Precedence Rules

- When unary operators have equal precedence, the operator on the right acts before the operation(s) on the left.
- Even when parentheses are not needed, they can be used to make the code clearer.

```
balance + (interestRate * balance)
```

- Spaces also make code clearer

```
balance + interestRate*balance
```

but spaces do not dictate precedence.

Sample Expressions

- Figure 2.3 Some Arithmetic Expressions in Java

Ordinary Math	Java (Preferred Form)	Java (Parenthesized)
$rate^2 + delta$	`rate * rate + delta`	`(rate * rate) + delta`
$2(salary + bonus)$	`2 * (salary + bonus)`	`2 * (salary + bonus)`
$\frac{1}{time + 3mass}$	`1 / (time + 3 * mass)`	`1 / (time + (3 * mass))`
$\frac{a-7}{t+9v}$	`(a - 7) / (t + 9 * v)`	`(a - 7) / (t + (9 * v))`

Specialized Assignment Operators

- Assignment operators can be combined with arithmetic operators (including `-`, `*`, `/`, and `%`, discussed later).

```
amount = amount + 5;
```

can be written as

```
amount += 5;
```

yielding the same results.

Case Study:Vending Machine Change

- Requirements
  - The user enters an amount between 1 cent and 99 cents.
  - The program determines a combination of coins equal to that amount.
  - For example, 55 cents can be two quarters and one nickel.

Case Study

- Sample dialog

```
Enter a whole number from 1 to 99.
The machine will determine a combination of coins.
87
87 cents in coins:
  3 quarters
  1 dime
  0 nickels
  2 pennies
```

Case Study

- Variables needed

```
int amount,
  quarters,
  dimes,
  nickels,
  pennies;
```

Case Study

- Algorithm - first version

```
1. Read the amount.
2. Find the maximum number of quarters
   in the amount.
3. Subtract the value of the quarters
   from the amount.
4. Repeat the last two steps for dimes,
   nickels, and pennies.
5. Print the original amount and the
   quantities of each coin.
```

Case Study,cont.

- The algorithm doesn't work properly
  - Original amount is changed by the intermediate steps.
  - Original value of `amount` is lost.
- Change the list of variables

```
int amount, originalAmount,
quarters, dimes, nickles, pennies;
```

  - Update the algorithm.

Case Study

- Algorithm – second version

```
1. Read the amount.
2. Make a copy of the amount.
3. Find the maximum number of quarters in
   the amount.
4. Subtract the value of the quarters
   from the amount.
5. Repeat the last two steps for dimes,
   nickels, and pennies.
6. Print the original amount and the
   quantities of each coin.
```

Case Study

- How do we determine the number of quarters (or dimes, nickels, or pennies) in an amount?
- There are 2 quarters in 55 cents, but there are also 2 quarters in 65 cents.
- That's because

```
55 / 2 = 2 and 65 / 25 = 2.
```

Case Study

- How do we determine the remaining amount?
- The remaining amount can be determined using the **`mod`** operator

```
55 % 25 = 5 and 65 % 25 = 15
```

- Similarly for dimes and nickels.
- Pennies are simply **`amount % 5.`**

Case Study

- The program should be tested with several different amounts.
- Test with values that give zero values for each possible coin denomination.
- Test with amounts close to
  - extreme values such as 0, 1, 98 and 99
  - coin denominations, such as 24, 25, and 26.

```
1. import java.util.Scanner;
2. public class ChangeMaker
3. {
4.     public static void main (String [] args)
5.     {
6.         int amount, originalAmount, quarters, dimes, nickels, pennies;
7.         System.out.println ("Enter a whole number from 1 to 99.");
8.         Scanner keyboard = new Scanner (System.in);
9.         amount = keyboard.nextInt();
10.        originalAmount = amount;
11.        quarters = amount / 25;
12.        amount = amount % 25;
13.        dimes = amount / 10;
14.        amount = amount % 10;
15.        nickels = amount / 5;
16.        amount = amount % 5;
17.        pennies = amount;
18.        System.out.println (originalAmount + " cents in coins can be given as:");
19.        System.out.println (quarters + " quarters");
20.        System.out.println (dimes + " dimes");
21.        System.out.println (nickels + " nickels and");
22.        System.out.println (pennies + " pennies");
23.    }
24.}
```

Increment and Decrement Operators

- Used to increase (or decrease) the value of a variable by 1
- Easy to use, important to recognize
- The increment operator

  `count++` or `++count`

- The decrement operator

  `count--` or `--count`

Increment and Decrement Operators

- equivalent operations

```
count++;
++count;
count = count + 1;

count--;
--count;
count = count - 1;
```

Increment and Decrement Operators in Expressions

- after executing

```
int m = 4;
int result = 3 * (++m)
```

`result` has a value of `15` and `m` has a value of `5`

- after executing

```
int m = 4;
int result = 3 * (m++)
```

`result` has a value of `12` and `m` has a value of `5`

The Class `String`

- We've used constants of type `String` already.

```
"Enter a whole number from 1 to 99."
```

- A value of type `String` is a
  - Sequence of characters
  - Treated as a single item.

The Empty String

- A string can have any number of characters, including zero.
- The string with zero characters is called the *empty* string.
- The empty string is useful and can be created in many ways including

```
String s3 = "";
```

The Unicode Character Set

- Most programming languages use the *ASCII* character set.
- Java uses the *Unicode* character set which includes the ASCII character set.
- The Unicode character set includes characters from many different alphabets (but you probably won't use them).

String Constants and Variables

- Declaring

```
String greeting;
```

```
greeting = "Hello!";
```

or

```
String greeting = "Hello!";
```

or

```
String greeting = new String("Hello!");
```

- Printing

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

Concatenation of Strings

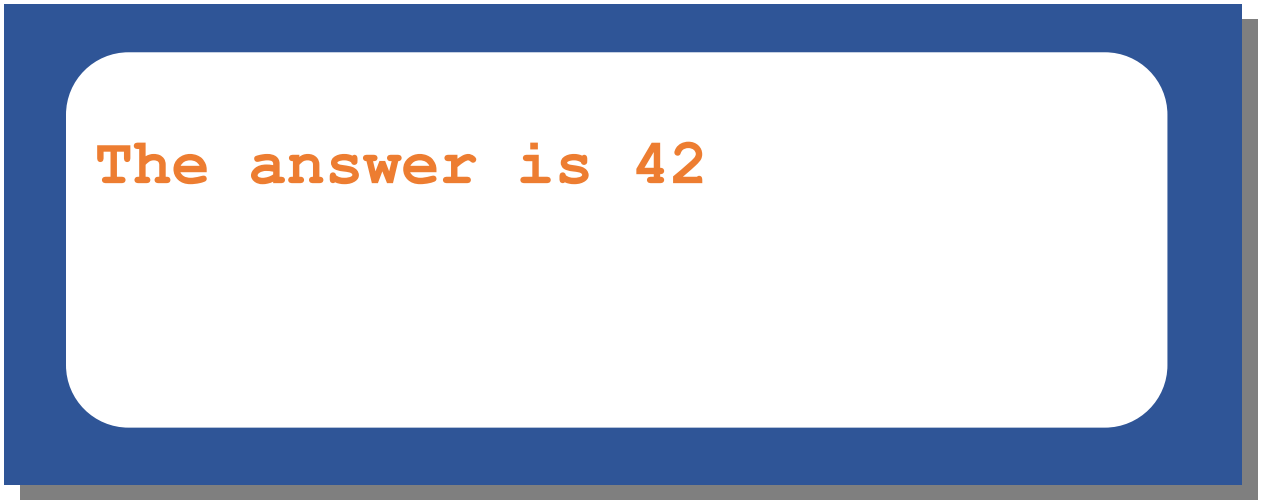
- Two strings are *concatenated* using the **+** operator.

```
String greeting = "Hello";
String sentence;
sentence = greeting + " officer";
System.out.println(sentence);
```

- Any number of strings can be concatenated using the **+** operator.

Concatenating Strings and Integers

```
String solution;
solution = "The answer is " + 42;
System.out.println (solution);
```



String Methods

- An object of the **`String`** class stores data consisting of a sequence of characters.
- Objects have methods as well as data
- The **`length()`** method returns the number of characters in a particular **`String`** object.

```
String greeting = "Hello";
int n = greeting.length();
```

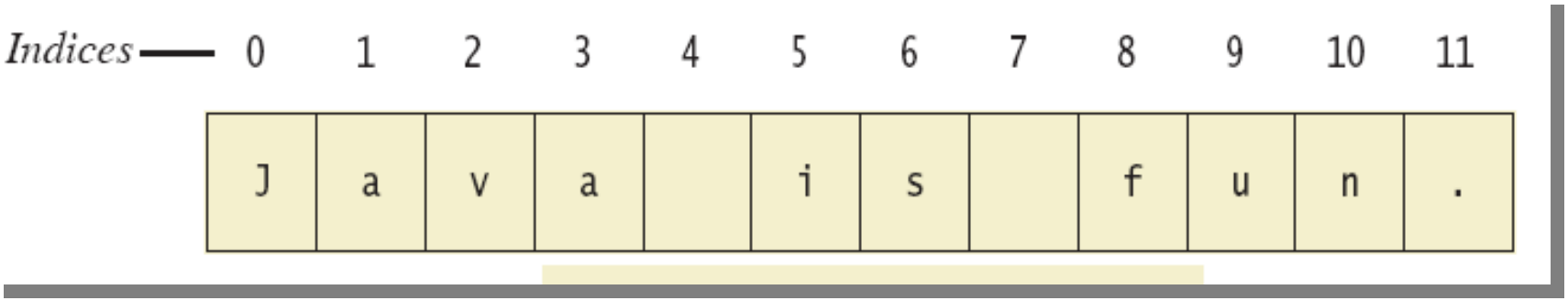
The Method `length()`

- The method `length()` returns an `int`.
- You can use a call to method `length()` anywhere an `int` can be used.

```
int count = command.length();
System.out.println("Length is " +
   command.length());
count = command.length() + 3;
```

`String` Indices

- Figure 2.4



- Positions start with 0, not 1.
  - The 'J' in "Java is fun." is in position 0
- A position is referred to an an *index.*
  - The `'f'` in `"Java is fun."` is at index 8.

FIGURE 2.5 Some Methods in the Class `String`

Method	Return Type	Example for `String s = "Java";`	Description
`charAt` (*index*)	`char`	`c = s.charAt(2);` `// c='v'`	Returns the character at *index* in the string. Index numbers begin at 0.
`compareTo` (a_*string*)	`int`	`i = s.compareTo("C++");` `// i is positive`	Compares this string with *a_string* to see which comes first in lexicographic (alphabetic, with upper before lower case) ordering. Returns a negative integer if this string is first, zero if the two strings are equal, and a positive integer if *a_string* is first.
`concat` (a_*string*)	`String`	`s2 = s.concat("rocks");` `// s2 = "Javarocks"`	Returns a new string with this string concatenated with *a_string*. You can use the + operator instead.
`equals` (a_*string*)	`boolean`	`b = s.equals("Java");` `// b = true`	Returns true if this string and *a_string* are equal. Otherwise returns false.
`equals IgnoreCase` (a_*string*)	`boolean`	`b = s.equals("Java");` `// b = true`	Returns true if this string and *a_string* are equal, considering upper and lower case versions of a letter to be the same. Otherwise returns false.
`indexOf` (a_*string*)	`int`	`i = s.indexOf("va");` `// i = 2`	Returns the index of the first occurrence of the substring *a_string* within this string or -1 if *a_string* is not found. Index numbers begin at 0.

`lastIndexOf` (a_*string*)	`int`	`i = s.lastIndexOf("a");` `// i = 3`	Returns the index of the last occurrence of the substring *a_string* within this string or -1 if *a_string* is not found. Index numbers begin at 0.
`length()`	`int`	`i = s.length();` `// i = 4`	Returns the length of this string.
`toLower Case()`	`String`	`s2 = s.toLowerCase();` `// s = "java"`	Returns a new string having the same characters as this string, but with any uppercase letters converted to lowercase. This string is unchanged.
`toUpper Case()`	`String`	`s2 = s.toUpperCase();` `// s2 = "JAVA"`	Returns a new string having the same characters as this string, but with any lowercase letters converted to uppercase. This string is unchanged.
`replace` (*oldchar, newchar)*	`String`	`s2 = s.replace('a','o');` `// s2 = "Jovo";`	Returns a new string having the same characters as this string, but with each occurrence of *oldchar* replaced by *newchar*.
`substring` (*start)*	`String`	`s2 = s.substring(2);` `// s2 = "va";`	Returns a new string having the same characters as the substring that begins at index *start* through to the end of the string. Index numbers begin at 0.
`substring` (*start,end*)	`String`	`s2 = s.substring(1,3);` `// s2 = "av";`	Returns a new string having the same characters as the substring that begins at index *start* through to but not including the character at index *end*. Index numbers begin at 0.

`trim( )`	`String`	`s = "  Java  ";`
		`s2 = s.trim();`
		`// s2 = "Java"`
			Returns a new string having the same characters as this string, but with leading and trailing whitespace removed.

```
1.public class StringDemo
2.{
3.    public static void main (String [] args)
4.    {
5.        String sentence = "Text processing is hard!";
6.        int position = sentence.indexOf ("hard");
7.        System.out.println (sentence);
8.        System.out.println ("01234567890123456789012 3");
9.        System.out.println ("The word \"hard\" starts at index " + position);
10.        sentence = sentence.substring (0, position) + "easy!";
11.        sentence = sentence.toUpperCase ();
12.        System.out.println ("The changed string is:");
13.        System.out.println (sentence);
14.    }
15.}
```

```
Text processing is hard!
01234567890123456789012 3
The word "hard" starts at index 19
The changed string is:
TEXT PROCESSING IS EASY!
```

Escape Characters

- How would you print

```
"Java" refers to a language.    ?
```

- The compiler needs to be told that the quotation marks (") do not signal the start or end of a string, but instead are to be printed.

```
System.out.println(
"\"Java\" refers to a language.");
```

Escape Characters

`\"`	Double quote.
`\'`	Single quote.
`\\`	Backslash.
`\n`	New line. Go to the beginning of the next line.
`\r`	Carriage return. Go to the beginning of the current line.
`\t`	Tab. Add whitespace up to the next tab stop.

- Figure 2.6
- Each escape sequence is a single character even though it is written with two symbols.

Examples

```
System.out.println("abc\\def");
```

```
abc\def
```

```
System.out.println("new\nline");
```

```
new
line
```

```
char singleQuote = '\'';
System.out.println
  (singleQuote);
```

```
'
```