Strings

Algoritma dan Pemrograman

Renny Pradina Kusumawardani, S.T., M.T., SCJP

Overview

- Penggunaan kelas String
- Penggunaan dokumentasi Java API
- Penggunaan kelas StringBuilder
- Operator numerik sisa hasil bagi
- Promosi dan casting variabel

Kelas String

- You have met Strings many times before….
- Diapit dengan tanda petik ganda (“ “)
  - setiap potong kode yang diapit dengan petik ganda akan diinterpretasikan sebagai tulisan harfiah, bukan variabel atau keyword
- Cara membuat objek bertipe data String:

  1. Langsung mendeklarasikan dan menginisiasi nilainya

  2. Menggunakan kata kunci “new”

     - menggunakan kata kunci “new” merupakan metode umum dalam instantiasi objek

Membuat Objek Bertipe Data String

1. Deklarasi dan inisiasi (pemberian nilai awal) secara langsung

```
String s = “Lalalala”;
```

2. Menggunakan kata kunci “new”

- `String s = new String();`

```
s = new String(“Aku senang sekali!”);
```

Kata Kunci “new”

- kata kunci “new” ini merupakan metode umum dalam pembuatan objek

- berfungsi untuk memanggil konstruktor dari suatu kelas

A Bit on Object-Oriented Programming

- Definisi program Java seluruhnya disusun dari kelas-kelas
- Kelas adalah deskripsi dari suatu tipe
- Secara umum, kelas adalah ‘cetak biru’ dari objek
- Contoh:

  Rumah di suatu kompleks

  - Pengembang memiliki cetak biru (*blue print*) dari rumah-rumah yang akan dibangun
  - Cetak biru != rumah

Kelas dan Objek

- Kelas adalah ‘cetak biru’ dari suatu tipe
- Objek adalah benda konkrit yang dibangun dengan mengacu kepada definisi kelas
- Q: apa arti kata ‘constructor’ secara harfiah?
- Kesimpulan….

Penggunaan Dokumentasi Java API

- API merupakan singkatan dari ‘Application Programming Interface’
  - Don’t worry about this. Sementara ini anda dapat memandang API sebagai kumpulan dari metode dan konstruktor yang dapat anda pergunakan langsung tanpa menulis kode untuk mendefinisikan metode atau konstruktor tersebut
  - Terdapat terlalu banyak kelas dan fungsionalitasnya yang tersedia untuk dihafalkan semua, sehingga anda **harus belajar membaca dokumentasi**
- Studi Kasus: Dokumentasi Java API untuk Kelas String

https://docs.oracle.com/javase/10/docs/api/java/lang/String.html

Beberapa Method String (1)

Berikut ini adalah beberapa method pada tipe data String yang perlu anda ingat:

- **length()** Returns the number of characters in a string
- **split()** Produces an array of parts of string, delimited by a character
- **substring()** Returns a part of a string
- **toLowerCase()** Returns a string, with uppercase characters converted to lowercase
- **toUpperCase()** Returns a string, with lowercase characters converted to uppercase
- **trim()** Removes whitespace from both ends of a string

Beberapa Method String (2)

Beberapa method tambahan

(catatan: diujikan pada Oracle Certified Associate, di samping metode-metode di atas)

- **charAt()** Returns the character located at the specified index
- **concat()** Appends one string to the end of another (+ also works)
- **equalsIgnoreCase()** Determines the equality of two strings, ignoring case
- **replace()** Replaces occurrences of a character with a new character
- **toString()** Returns the value of a string

Latihan 5 Menit

- Lihat dokumentasi Java API untuk String
- Lihat konstruktor apa saja yang ada
- Lihat metode-metode yang telah disebutkan.
  - Apa fungsi dari masing-masing metode tersebut?
  - Apa tipe data yang dikembalikan oleh masing-masing metode?

Method dan Penggunaannya

- Method ada yang bersifat static, ada yang merupakan instance method
  - Ciri-cirinya: yang bersifat static nama methodnya didahului dengan kata kunci `static`
- Method static dapat dipergunakan dengan referensi terhadap kelas (dan dapat juga dengan referensi terhadap objek)
- Instance method hanya bisa dipergunakan dengan referensi terhadap objek

Sintaks Penggunaan Method (1)

- How to refer to object / class?
  - menggunakan operator dot (‘.’)

- Contoh 1:

mendapatkan panjang dari suatu String dengan menggunakan fungsi `int length()`:

```
String s = “Oooooooooooooo… begitu ya.”;
```

Untuk mendapatkan panjang dari string s di atas: **`s.length()`**

(Silakan mencoba: `System.out.println(s.length());`)

Sintaks Penggunaan Method (2)

- Contoh 2:

mendapatkan versi dari String s yang seluruhnya terdiri dari huruf kapital dengan menggunakan `String toUpperCase()`:

```
s.toUpperCase()
```

Silakan mencoba:

```
System.out.println(s.toUpperCase());
```

Aktivitas Belajar Terstruktur

- Silakan mencoba di rumah metode-metode pada kelas String yang disebutkan di atas

Strings are Immutable

- Penting!!! Silakan mencoba:

```
String s = “Ooooooooooooooo… begitu ya.”;

System.out.println(s);

System.out.println(s.toUpperCase());

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

- Apa yang anda amati?

StringBuilder

- Use StringBuilder instead of StringBuffer

Objects and classes

- **object:** An entity that contains:
  - *data* (variables), and
  - *behavior* (methods).

- **class**: A program, or a type of objects.

- Examples:
  - The class `String` represents objects that store text.
  - The class `DrawingPanel` represents graphical window objects.
  - The class `Scanner` represents objects that read information from the keyboard, files, and other sources.

Strings

- **string**: An object storing a sequence of text characters.
  - Unlike most other objects, a `String` is not created with `new`.

```
String name = "text";
String name = expression;
```

- Examples:

```
String name = "Marla Singer";
int x = 3;
int y = 5;
String point = "(" + x + ", " + y + ")";
```

Indexes

- Characters of a string are numbered with 0-based *indexes*:

```
String name = "P. Diddy";
```

index	0	1	2	3	4	5	6	7
char	P	.		D	i	d	d	y

- The first character's index is always 0
- The last character's index is 1 less than the string's length

- The individual characters are values of type `char` (seen later)

`String` methods

Method name	Description
`indexOf(`**str**`)`	index where the start of the given string appears in this string (-1 if it is not there)
`length()`	number of characters in this string

`substring(`**index1**`,` **index2**`)`
or
`substring(`**index1**`)`
	the characters in this string from *index1* (inclusive) to *index2* (exclusive);
	if *index2* omitted, grabs till end of string

`toLowerCase()`	a new string with all lowercase letters
`toUpperCase()`	a new string with all uppercase letters

- These methods are called using the dot notation:

```
String gangsta = "Dr. Dre";
System.out.println(gangsta.length());   // 7
```

`String` method examples

```
//      index 012345678901
String s1 = "Stuart Reges";
String s2 = "Marty Stepp";
System.out.println(s1.length());           // 12
System.out.println(s1.indexOf("e"));       // 8
System.out.println(s1.substring(7, 10))    // "Reg"

String s3 = s2.substring(2, 8);
System.out.println(s3.toLowerCase());      // "rty st"
```

- Given the following string:

```
//          index 012345678901234567890
String book = "Building Java Programs";
```

  - How would you extract the word `"Java"` ?
  - How would you extract the first word from any string?

Modifying strings

- Methods like `substring`, `toLowerCase`, etc. create/return a new string, rather than modifying the current string.

```
String s = "lil bow wow";
s.toUpperCase();
System.out.println(s);   // lil bow wow
```

- To modify a variable, you must reassign it:

```
String s = "lil bow wow";
s = s.toUpperCase();
System.out.println(s);   // LIL BOW WOW
```

Strings as parameters

```
public class StringParameters {
    public static void main(String[] args) {
        sayHello("Marty");

        String teacher = "Helene";
        sayHello(teacher);
    }

    public static void sayHello(String name) {
        System.out.println("Welcome, " + name);
    }
}
```

Output:

```
Welcome, Marty
Welcome, Helene
```

Strings as user input

- `Scanner`'s `next` method reads a word of input as a `String`.

```
Scanner console = new Scanner(System.in);
System.out.print("What is your name? ");
String name = console.next();
name = name.toUpperCase();
System.out.println(name + " has " + name.length() +
    " letters and starts with " + name.substring(0, 1));
```

Output:

```
What is your name? Madonna
MADONNA has 7 letters and starts with M
```

- The `nextLine` method reads a line of input as a `String`.

```
System.out.print("What is your address? ");
String address = console.nextLine();
```

Comparing strings

- Relational operators such as < and == fail on objects.

```
Scanner console = new Scanner(System.in);
System.out.print("What is your name? ");
String name = console.next();
if (name == "Barney") {
    System.out.println("I love you, you love me,");
    System.out.println("We're a happy family!");
}
```

- This code will compile, but it will not print the song.

- == compares objects by *references* (seen later), so it often gives `false` even when two `String`s have the same letters.

The `equals` method

- Objects are compared using a method named `equals`.

```
Scanner console = new Scanner(System.in);
System.out.print("What is your name? ");
String name = console.next();
if (name.equals("Barney")) {
    System.out.println("I love you, you love me,");
    System.out.println("We're a happy family!");
}
```

- Technically this is a method that returns a value of type `boolean`, the type used in logical tests.

`String` test methods

Method	Description
`equals(`**str**`)`	whether two strings contain the same characters
`equalsIgnoreCase(`**str**`)`	whether two strings contain the same characters, ignoring upper vs. lower case
`startsWith(`**str**`)`	whether one contains other's characters at start
`endsWith(`**str**`)`	whether one contains other's characters at end
`contains(`**str**`)`	whether the given string is found within this one

```
String name = console.next();
if (name.startsWith("Dr.")) {
    System.out.println("Are you single?");
} else if (name.equalsIgnoreCase("LUMBERG")) {
    System.out.println("I need your TPS reports.");
}
```

Strings question

- Write a program that reads a person's name and converts it into a "gangsta name."

Output (run 1):

```
Type your name, playa: Peter Griffin
(M)ale or (F)emale? m
Your gangsta name is "P. GRIFFIN Daddy Peter-izzle"
```

Output (run 2):

```
Type your name, playa: Marge Simpson
(M)ale or (F)emale? F
Your gangsta name is "M. SIMPSON Goddess Marge-izzle"
```

Strings answer

```
// This program prints your "gangsta" name.
import java.util.*;

public class GangstaName {
    public static void main(String[] args) {
        Scanner console = new Scanner(System.in);
        System.out.print("Type your name, playa: ");
        String name = console.nextLine();

        System.out.print("(M)ale or (F)emale: ");
        String gender = console.next();

        // split name into first/last name and initials
        String first = name.substring(0, name.indexOf(" "));
        String last = name.substring(name.indexOf(" ") + 1);
        last = last.toUpperCase();
        String fInitial = first.substring(0, 1);

        String title;
        if (gender.equalsIgnoreCase("m")) {
            title = "Daddy";
        } else {
            title = "Goddess";
        }

        System.out.println("Your gangsta name is \"" + fInitial + ". "
                + last + " " + title + " " + first + "-izzle\"");
    }
}
```

Type `char`

- `char` : A primitive type representing single characters.
  - Each character inside a `String` is stored as a `char` value.
  - Literal `char` values are surrounded with apostrophe (single-quote) marks, such as `'a'` or `'4'` or `'\n'` or `'\''`

  - It is legal to have variables, parameters, returns of type `char`

```
char letter = 'S';
System.out.println(letter);                    // S
```

- `char` values can be concatenated with strings.

```
char initial = 'P';
System.out.println(initial + " Diddy");  // P Diddy
```

The `charAt` method

- The `char`s in a `String` can be accessed using the `charAt` method.

```
String food = "cookie";
char firstLetter = food.charAt(0);   // 'c'

System.out.println(firstLetter + " is for " + food);
System.out.println("That's good enough for me!");
```

- You can use a `for` loop to print or examine each character.

```
String major = "CSE";
for (int i = 0; i < major.length(); i++) {
    char c = major.charAt(i);
    System.out.println(c);
}
```

Output:

```
C
S
E
```

`char` vs. `int`

- All `char` values are assigned numbers internally by the computer, called *ASCII* values.
  - Examples:

    `'A'` is 65, `'B'` is 66, `' '` is 32

    `'a'` is 97, `'b'` is 98, `'*'` is 42

- Mixing `char` and `int` causes automatic conversion to `int`.

  `'a' + 10` is 107, `'A' + 'A'` is 130

- To convert an `int` into the equivalent `char`, type-cast it.

  `(char) ('a' + 2)` is `'c'`

`char` vs. `String`

- `"h"` is a `String`
  `'h'` is a `char` (the two behave differently)

- `String` is an object; it contains methods

```
String s = "h";
s = s.toUpperCase();          // 'H'
int len = s.length();         //  1
char first = s.charAt(0);     // 'H'
```

- `char` is primitive; you can't call methods on it

```
char c = 'h';
c = c.toUpperCase();    // ERROR: "cannot be dereferenced"
```

  - What is `s + 1`? What is `c + 1`?
  - What is `s + s`? What is `c + c`?

Comparing `char` values

- You can compare `char` values with relational operators:

```
'a' < 'b'   and   'X' == 'X'   and   'Q' != 'q'
```

  - An example that prints the alphabet:

```
for (char c = 'a'; c <= 'z'; c++) {
    System.out.print(c);
}
```

- You can test the value of a string's character:

```
String word = console.next();
if (word.charAt(word.length() - 1) == 's') {
    System.out.println(word + " is plural.");
}
```

`String/char` question

- A *Caesar cipher* is a simple encryption where a message is encoded by shifting each letter by a given amount.
  - e.g. with a shift of 3, A → D, H → K, X → A, and Z → C

- Write a program that reads a message from the user and performs a Caesar cipher on its letters:

```
Your secret message: Brad thinks Angelina is cute
Your secret key: 3
The encoded message: eudg wklqnv dqjholqd lv fxwh
```

Strings answer 1

```
// This program reads a message and a secret key from the user and
// encrypts the message using a Caesar cipher, shifting each letter.

import java.util.*;

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

        System.out.print("Your secret message: ");
        String message = console.nextLine();
        message = message.toLowerCase();

        System.out.print("Your secret key: ");
        int key = console.nextInt();

        encode(message, key);
    }

    ...
```

Strings answer 2

```
// This method encodes the given text string using a Caesar
// cipher, shifting each letter by the given number of places.
public static void encode(String text, int shift) {
    System.out.print("The encoded message: ");
    for (int i = 0; i < text.length(); i++) {
        char letter = text.charAt(i);

        // shift only letters (leave other characters alone)
        if (letter >= 'a' && letter <= 'z') {
            letter = (char) (letter + shift);

            // may need to wrap around
            if (letter > 'z') {
                letter = (char) (letter - 26);
            } else if (letter < 'a') {
                letter = (char) (letter + 26);
            }
        }
        System.out.print(letter);
    }
    System.out.println();
}
}
```

Strings

- Java string is a sequence of characters. They are objects of type String.
- Once a String object is created it cannot be changed. Stings are Immutable.
- To get changeable strings use the class called StringBuffer.
- String and StringBuffer classes are declared final, so there cannot be subclasses of these classes.
- The default constructor creates an empty string.

```
String s = new String();
```

Creating Strings

- `String str = "abc";` is equivalent to:

```
char data[] = {'a', 'b', 'c'};
String str = new String(data);
```

- If data array in the above example is modified after the string object str is created, then str remains unchanged.
- Construct a string object by passing another string object.

  String str2 = new String(str);

String Operations

- The length() method returns the length of the string.
  Eg: System.out.println("Hello".length()); // prints 5
- The + operator is used to concatenate two or more strings.
  Eg: String myname = "Harry"
  String str = "My name is" + myname+ ".";
- For string concatenation the Java compiler converts an operand to a String whenever the other operand of the + is a String object.

String Operations

- Characters in a string can be extracted in a number of ways.

public char **charAt**(int index)

  – Returns the character at the specified index. An index ranges from 0 to length() - 1. The first character of the sequence is at index 0, the next at index 1, and so on, as for array indexing.

```
char ch;
ch = "abc".charAt(1); // ch = "b"
```

String Operations

- **getChars() -** Copies characters from this string into the destination character array.

  ```
  public void getChars(int srcBegin, int srcEnd,
  char[] dst, int dstBegin)
  ```

  - srcBegin - index of the first character in the string to copy.
  - srcEnd - index after the last character in the string to copy.
  - dst - the destination array.
  - dstBegin - the start offset in the destination array.

String Operations

- **equals() -** Compares the invoking string to the specified object. The result is true if and only if the argument is not null and is a String object that represents the same sequence of characters as the invoking object.

  ```
  public boolean equals(Object anObject)
  ```

- **equalsIgnoreCase()-** Compares this String to another String, ignoring case considerations. Two strings are considered equal ignoring case if they are of the same length, and corresponding characters in the two strings are equal ignoring case.

  ```
  public boolean equalsIgnoreCase(String
  anotherString)
  ```

String Operations

- **startsWith()** – Tests if this string starts with the specified prefix.

```
public boolean startsWith(String prefix)
"Figure".startsWith("Fig"); // true
```

- **endsWith() -** Tests if this string ends with the specified suffix.

```
public boolean endsWith(String suffix)
"Figure".endsWith("re"); // true
```

String Operations

- **startsWith()** -Tests if this string starts with the specified prefix beginning at a specified index.

```
public boolean startsWith(String prefix,
int toffset)

  prefix - the prefix.
      toffset - where to begin looking in the
      string.

"figure".startsWith("gure", 2); // true
```

String Operations

- **compareTo() -** Compares two strings lexicographically.
  - The result is a negative integer if this String object lexicographically precedes the argument string.
  - The result is a positive integer if this String object lexicographically follows the argument string.
  - The result is zero if the strings are equal.
  - compareTo returns 0 exactly when the equals(Object) method would return true.

```
public int compareTo(String anotherString)
public int compareToIgnoreCase(String str)
```

String Operations

indexOf – Searches for the first occurrence of a character or substring. Returns -1 if the character does not occur.

`public int` **`indexOf`**`(int ch)` – Returns the index within this string of the first occurrence of the specified character.

`public int` **`indexOf`**`(String str)` - Returns the index within this string of the first occurrence of the specified substring.

```
String str = "How was your day today?";
str.indexof('t');
str("was");
```

String Operations

`public int` **`indexOf`**`(int ch, int fromIndex)` - Returns the index within this string of the first occurrence of the specified character, starting the search at the specified index.

`public int` **`indexOf`**`(String str, int fromIndex)` - Returns the index within this string of the first occurrence of the specified substring, starting at the specified index.

String str = “How was your day today?”;
str.indexof(‘a’, 6);
str(“was”, 2);

String Operations

lastIndexOf() –Searches for the last occurrence of a character or substring. The methods are similar to indexOf().

substring() - Returns a new string that is a substring of this string. The substring begins with the character at the specified index and extends to the end of this string.

```
public String substring(int beginIndex)
```

Eg: `"unhappy".substring(2) returns "happy"`

String Operations

- `public String` **`substring`**`(int beginIndex, int endIndex)`

  `Eg: "smiles".substring(1, 5)`

  `returns "mile“`

String Operations

concat() - Concatenates the specified string to the end of this string.

If the length of the argument string is 0, then this String object is returned.

Otherwise, a new String object is created, containing the invoking string with the contents of the str appended to it.

```
public String concat(String str)
"to".concat("get").concat("her") returns
   "together"
```

String Operations

- replace()- Returns a new string resulting from replacing all occurrences of oldChar in this string with newChar.

```
public String replace(char oldChar, char newChar)
```

```
"mesquite in your cellar".replace('e', 'o')
   returns "mosquito in your collar"
```

String Operations

- **trim() -** Returns a copy of the string, with leading and trailing whitespace omitted.

```
public String trim()

String s = "  Hi Mom!  ".trim();
S = "Hi Mom!"
```

- **valueOf() –** Returns the string representation of the char array argument.

```
public static String valueOf(char[] data)
```

String Operations

- The contents of the character array are copied; subsequent modification of the character array does not affect the newly created string.

Other forms are:

```
public static String valueOf(char c)
public static String valueOf(boolean b)
public static String valueOf(int i)
public static String valueOf(long l)
public static String valueOf(float f)
public static String valueOf(double d)
```

String Operations

- **toLowerCase():** Converts all of the characters in a String to lower case.
- **toUpperCase():** Converts all of the characters in this String to upper case.

```
public String toLowerCase()
public String toUpperCase()

Eg: "HELLO THERE".toLowerCase();
    "hello there".toUpperCase();
```

StringBuffer

- A StringBuffer is like a String, but can be modified.
- The length and content of the StringBuffer sequence can be changed through certain method calls.
- StringBuffer defines three constructors:
  - StringBuffer()
  - StringBuffer(int size)
  - StringBuffer(String str)

StringBuffer Operations

- The principal operations on a StringBuffer are the append and insert methods, which are overloaded so as to accept data of any type.

Here are few append methods:

```
StringBuffer append(String str)
StringBuffer append(int num)
```

- The append method always adds these characters at the end of the buffer.

StringBuffer Operations

- The insert method adds the characters at a specified point.

Here are few insert methods:

```
StringBuffer insert(int index, String str)
StringBuffer append(int index, char ch)
```

Index specifies at which point the string will be inserted into the invoking StringBuffer object.

StringBuffer Operations

- **delete() -** Removes the characters in a substring of this StringBuffer. The substring begins at the specified start and extends to the character at index end - 1 or to the end of the StringBuffer if no such character exists. If start is equal to end, no changes are made.

```
public StringBuffer delete(int start, int end)
```

StringBuffer Operations

- **replace() -** Replaces the characters in a substring of this StringBuffer with characters in the specified String.

```
public StringBuffer replace(int start, int end,
   String str)
```

- **substring() -** Returns a new String that contains a subsequence of characters currently contained in this StringBuffer. The substring begins at the specified index and extends to the end of the StringBuffer.

```
public String substring(int start)
```

StringBuffer Operations

- **reverse() -** The character sequence contained in this string buffer is replaced by the reverse of the sequence.

```
public StringBuffer reverse()
```

- **length() -** Returns the length of this string buffer.

```
public int length()
```

StringBuffer Operations

- **capacity() -** Returns the current capacity of the String buffer. The capacity is the amount of storage available for newly inserted characters.

```
public int capacity()
```

- **charAt() -** The specified character of the sequence currently represented by the string buffer, as indicated by the index argument, is returned.

```
public char charAt(int index)
```

StringBuffer Operations

- **getChars() -** Characters are copied from this string buffer into the destination character array dst. The first character to be copied is at index srcBegin; the last character to be copied is at index srcEnd-1.

```
public void getChars(int srcBegin, int srcEnd,
char[] dst, int dstBegin)
```

- **setLength() -** Sets the length of the StringBuffer.

```
public void setLength(int newLength)
```

Examples: StringBuffer

```
StringBuffer sb = new StringBuffer(“Hello”);
  sb.length(); // 5
  sb.capacity(); // 21 (16 characters room is
                    added if no size is specified)
  sb.charAt(1); // e
  sb.setCharAt(1,’i’); // Hillo
  sb.setLength(2); // Hi
  sb.append(“l”).append(“l”); // Hill
  sb.insert(0, “Big “); // Big Hill
```

Examples: StringBuffer

```
sb.replace(3, 11, “”); // Big
sb.reverse(); // gib
```

Strings

- Strings are fundamental part of all computing languages.
- At the basic level, they are just a data structure that can hold a series of characters
- However, strings are not implemented as a character array in Java as in other languages.

Strings in Java

- Strings are implemented as two classes in Java
- java.lang.String provides an unchangeable String object
- java.lang.StringBuffer provides a String object that can be amended

Basic String Methods

- length() returns the length of the string
- toLowerCase() converts the string to lower case
- toUpperCase() converts the string to upper case
- replace(char, char) replaces occurrences of one character with another character

Basic Strings continued

- Basic strings are not meant to change frequently so there are no add or append methods
- However the concat(String) method does allow two strings to be concatenated together

Basic Strings continued

- Substrings of a String object can also be accessed
- A portion of String object can be copied to a character array using the getChars() method
- The substring() method can return substring beginning at a specified offset

Searching a string

- Methods for searching strings
  - indexOf(*x*) searches for the first occurrence of *x*
  - indexOf(*x*, *y*) searches for the first occurrence of *x* after the offset of y
  - lastIndexOf(*x*) searches backwards for the first occurrence of *x*
  - lastIndexOf(*x*, *y*) searches backwards for the first occurrence of *x* after the offset of y

Example of string search

- indexOf(*x*) and indexOf(*x*, *y*) can find all occurrences of a character(s) in a string

```
public void paint(Graphics g) {
 String str = new String("Wish You Were Here");
      int count = 0;
      int fromIndex = 0;
      while(fromIndex != -1) {
           fromIndex = str.indexOf("er", fromIndex);
       if (fromIndex != -1) {
           count++;
           fromIndex++;
      }
   }
g.drawString(String.valueOf(count), 10, 10); }
```

Parsing Strings

- Strings can be parsed with the StringTokenizer class
- The default delimiters (space, tab, newline and carriage return) can be used for parsing sentences
- By specifying different delimiters, a wide variety of strings may be parsed

Parsing Strings continued

- Different default constructors are provided
  - Tokenize the string based on the default delimiters
  - Tokenize the string based on a specified set of delimiters
  - Tokenize the string based on a specified set of delimiters with a boolean flag to specify whether the delimiters should also be returned as tokens

StringBuffer class

- The StringBuffer class is provided for strings that need may need to be changed
- The StringBuffer class contains methods for both inserting and appending text
- An object created as a StringBuffer can easily be converted to an object of the String class if needed

More on StringBuffer Class

- Conversion may be needed because many Java library methods expect a string
- The toString() method is used for converting a StringBuffer object to a String object
- Example of converting a StringBuffer to a String:

```
public void paint(Graphics g) {
   StringBuffer buf = new StringBuffer("Hello, World");
   g.drawString(buf.toString(), 10, 10);
}
```

More on StringBuffer Class

- StringBuffer objects are mutable and capacity & length affect performance
- If the StringBuffer object needs to be expanded during an append or insert, a new array is created and the old data copied to it
- Use capacity() and ensureCapacity(int) methods to minimize expansions

Length v. Capacity

- The length() method returns the length of the string in the StringBuffer
- The capacity() method returns the total “space” in a StringBuffer
- The ensureCapacity(int) method insures the StringBuffer has at least the specified amount of capacity remaining

Length v. Capacity con't

- Examples of length() and capacity() methods

```
StringBuffer buf = new StringBuffer(25);//
   creates StringBuffer with length 25
buf.append("13 Characters"); // appends 13
   characters
int len = buf.length(); // length() returns 13
int cap = buf.capacity(); // capacity returns 25
```

Operator Numerik Sisa Hasil Bagi

- Operator sisa hasil bagi biasanya disebut dengan modulo (‘%’)
- Biasanya dipergunakan pada nilai integer, untuk melakukan pengulangan setiap nilai tertentu
- Contoh:

```
for (int i = 0; i < 100; i++) {
  if (i%3==0) {
    System.out.println(i + “ adalah angka kelipatan 3”);
  }
}
```

Promosi dan Casting

- https://docs.oracle.com/javase/specs/jls/se7/html/jls-5.html

1. int + double = ?
2. double + String = ?
3.
```
int a = 10;

double b = a;
```
4. Bagaimana dengan:

```
double a = 9.0;

int b = a + 1;
```

Casting Implisit dan Eksplisit

- Pada dasarnya, Java adalah bahasa pemrograman yang strongly-typed (strong-typing v.s. weak-typing: https://en.wikipedia.org/wiki/Strong_and_weak_typing)

- Jika tipe data tidak sesuai dengan yang diharapkan, akan terjadi compiler error

- Contoh: pada contoh 4 di depan

- Namun, pada kasus di mana konversi tidak mengurangi presisi data, casting (penyesuaian) ke tipe data yang lebih 'lebar' (membutuhkan memori yang lebih banyak) akan terjadi secara otomatis

—> **implicit casting**! —> contoh: pada contoh 3

Sintaks Casting Explicit

- Pengubahan tipe data masih dimungkinkan dengan explicit-casting

- Sintaks:

```
(tipe_data_tujuan) variabel_yang_di_casting
```

- Contoh 4:

```
double a = 9.0;

int b = (int)a + 1;
```

Casting Explisit pada Objek

- Casting tidak hanya bisa dilakukan pada tipe data primitif, namun juga pada data bertipe objek

- Contoh:

```
Animal aCat   = new Animal();

Cat    miaow = (Cat)aCat;
```

- Ada persyaratan bagaimana ini bisa dilakukan, but don't worry about it just yet since we haven't really talked about objects :-)