Revised: 8/23/2025

Text Processing

Michael C. Hackett

Associate Professor, Computer Science

COMMUNITY COLLEGE OF PHILADELPHIA

Lecture Topics

- Character Class
- String methods for processing text
- StringBuilders
- Tokenization
- CSV Files

Colors/Fonts

- Local Variable Names – Brown
- Primitive data types – Fuchsia
- Literals – Blue
- Keywords – Orange
- Object names – Green
- Operators/Punctuation – Black
- Field Names – Lt Blue
- Method Names – Purple
- Parameter Names – Gold
- Comments – Gray
- Package Names – Pink

Source Code – **Consolas**
Output – Courier New

Boolean expression is false

Boolean expression is true

Wrapper Classes

- A wrapper class is an object/class that holds the value of a primitive data type.
- A wrapper class is NOT a primitive; Wrapper classes allows the a primitive to act like an object.
- Wrapping a primitive in an object allows methods to manipulate the data contained within it.

Wrapper Classes in Java

Wrapper Class	Associated Primitive
Byte	byte
Short	short
Integer*	int
Long	long
Float	float
Double*	double
Character	char
Boolean	boolean

*- We have been using these wrappers for parsing integer and double values from Strings: Double.parseDouble() and Integer.parseInt()

Character Class

- A wrapper class for the primitive data type char.

- You can use its static methods without instantiating it (like Integer and Double)

Character Class – Testing if a char is a letter

```
char alphaChar = 'm';
char numberChar = '6';
char symbolChar = '!';
char blankChar = ' ';

Character.isLetter(alphaChar); //Returns TRUE
Character.isLetter(numberChar); //Returns FALSE
Character.isLetter(symbolChar); //Returns FALSE
Character.isLetter(blankChar); //Returns FALSE
```

Character Class – Testing if a char is a digit

```
char alphaChar = 'm';
char numberChar = '6';
char symbolChar = '!';
char blankChar = ' ';

Character.isDigit(alphaChar); //Returns FALSE
Character.isDigit(numberChar); //Returns TRUE
Character.isDigit(symbolChar); //Returns FALSE
Character.isDigit(blankChar); //Returns FALSE
```

Character Class – Testing if a char is a letter or digit

```
char alphaChar = 'm';
char numberChar = '6';
char symbolChar = '!';
char blankChar = ' ';

Character.isLetterOrDigit(alphaChar); //Returns TRUE
Character.isLetterOrDigit(numberChar); //Returns TRUE
Character.isLetterOrDigit(symbolChar); //Returns FALSE
Character.isLetterOrDigit(blankChar); //Returns FALSE
```

Character Class – Testing if a char is uppercase

```
char alphaChar = 'm';
char numberChar = '6';
char symbolChar = '!';
char blankChar = ' ';

Character.isUpperCase(alphaChar); //Returns FALSE
Character.isUpperCase(numberChar); //Returns FALSE
Character.isUpperCase(symbolChar); //Returns FALSE
Character.isUpperCase(blankChar); //Returns FALSE
```

Character Class – Testing if a char is lowercase

```
char alphaChar = 'm';
char numberChar = '6';
char symbolChar = '!';
char blankChar = ' ';

Character.isLowerCase(alphaChar); //Returns TRUE
Character.isLowerCase(numberChar); //Returns FALSE
Character.isLowerCase(symbolChar); //Returns FALSE
Character.isLowerCase(blankChar); //Returns FALSE
```

Character Class – Testing if a char is lowercase

```
char alphaChar = 'm';
char numberChar = '6';
char symbolChar = '!';
char blankChar = ' ';

Character.isWhitespace(alphaChar); //Returns FALSE
Character.isWhitespace(numberChar); //Returns FALSE
Character.isWhitespace(symbolChar); //Returns FALSE
Character.isWhitespace(blankChar); //Returns TRUE
```

Character Class – Converting a char to uppercase

```
char alphaChar = 'm';

Character.toUpperCase(alphaChar); //Returns 'M'
```

Character Class – Converting a char to lowercase

```
char alphaChar = 'P';

Character.toLowerCase(alphaChar); //Returns 'p'
```

String Character Indexes

- Characters in a string are indexed. Spaces count!
- Indexing begins at zero.

```
String example = "Example String";
```

E	x	a	m	p	l	e		S	t	r	i	n	g
0	1	2	3	4	5	6	7	8	9	10	11	12	13

- This String is 14 characters long (0-13)

Determining what character is at an index

- The charAt method gets the character at a certain index of a String.
- Takes one parameter (an int): The desired index

```
String test = "This is a test.";
char selectedChar = test.charAt(5);
System.out.println(selectedChar);
```

Output:

```
i
```

StringIndexOutOfBoundsException

- A StringIndexOutOfBoundsException occurs when you try to access an index that is out of range.

```
String letters = "abcd";
char myChar = letters.charAt(4);
System.out.println(myChar);
```

```
Exception in thread "main" java.lang.StringIndexOutOfBoundsException:
String index out of range: 4
at java.lang.String.charAt(Unknown Source)
at test2.main(test2.java:8)
```

Getting a substring (with one parameter)

- A ***substring*** is a portion of a String.
- The substring method allows you to get a substring.
- One parameter (an int) - The starting index.

```
String test = "This is a test.";
String testSubstring = test.substring(8);
System.out.println(testSubstring);
```

```
a test.
```

The substring method with one parameter returns a substring that starts with, and includes, the supplied starting index.

Getting a substring (two parameters)

- A second substring method (same name) allows you to get a substring using a starting and ending index.
- Two parameter (both ints)- First is the starting index, second is the ending index.
- The character at the ending index is NOT INCLUDED!

```
String test = "This is a test.";
String testSubstring = test.substring(0, 3);
System.out.println(testSubstring);
```

```
Thi
```

The substring method with two parameters returns a substring that starts with, and includes, the supplied starting index up to but EXCLUDING the ending index.

String Length

- The length method gets the number of characters in a String.
- Spaces count.
- Takes no parameters.

```
String test = "This is a test.";
int exampleLength = test.length();
System.out.println(exampleLength);
```

```
15
```

Replacing parts of a String

- The replace method is an easy way to replace part of a String with new data.
- Two parameters (both Strings)- first is the String to find, second is what to replace it with. *CASE SENSITIVE*

```
String origString = "Today is Monday.";
String newString = origString.replace("Monday", "Tuesday");
System.out.println(newString);
```

```
Today is Tuesday.
```

Note the value of the String variable origString **does not change**.
The replace method returns a new String with the every sequence of the first parameter replaced with the second parameter.

Replacing parts of a String

- Be careful. All matches will be replaced!

```
String origString = "Today is Monday.";
String newString = origString.replace("day", "night");
System.out.println(newString);
```

```
Tonight is Monnight.
```

Note the value of the String variable origString **does not change**.
The replace method returns a new String with the every sequence of the first parameter replaced with the second parameter.

StringBuilders

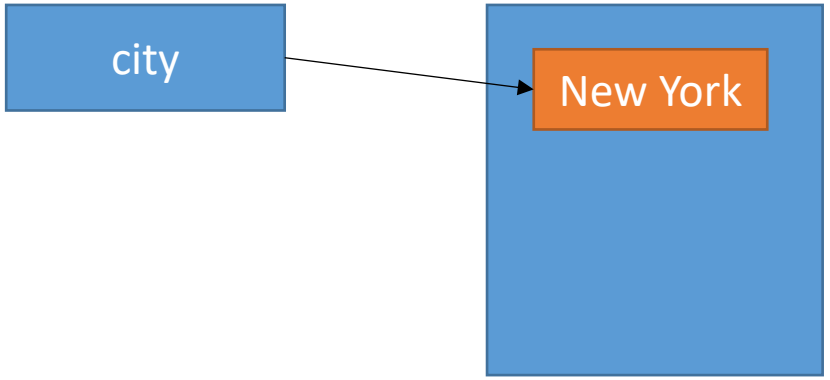
- The StringBuilder class is similar to a String in many regards.

- The biggest difference is that Strings are immutable and StringBuilders are mutable.

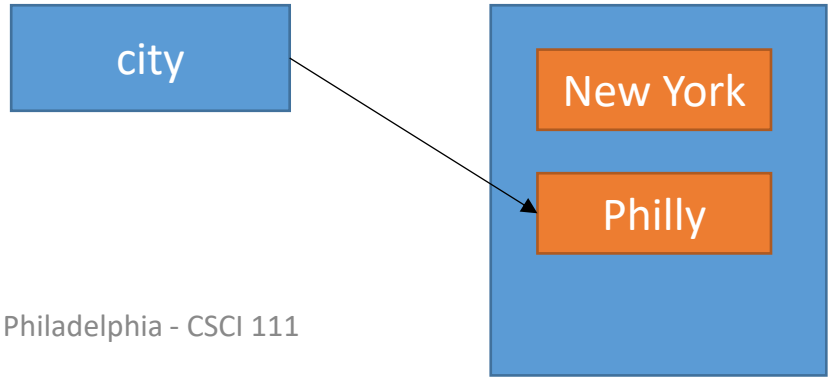
Strings

- When you change the value of a String, your String just points to a new value in the String pool.

```
String city;
city = "New York";
```




```
city = "Philly";
```



"New York" will eventually be picked up and destroyed by the JVM's garbage collector

Strings vs StringBuilders

- If your String will not be altered much (or at all) in your program, use a String.

- If your String will be reused and/or altered frequently in your program, use a StringBuilder.
  - For instance, StringBuilders are more efficient at concatenation.

Creating a StringBuilder object

```
StringBuilder sbExample1 = new StringBuilder();
```

- Creates an empty StringBuilder with enough memory for 16 characters.
  - The is just a default value; StringBuilders grow in memory size automatically.

Creating a StringBuilder object

```
StringBuilder sbExample2 = new StringBuilder(25);
```

- Creates an empty StringBuilder with space for a String 25 characters.
  - Again, the 25 length is not permanent; StringBuilders grow in memory size automatically.

Creating a StringBuilder object

```
StringBuilder sbExample3 = new StringBuilder("Starting string");
```

- Creates a StringBuilder with the provided String.
  - The actual amount of memory space it takes up is the length of the string plus 16 characters.

Appending to a StringBuilder

- A StringBuilder's overloaded append method can take in a variety of types.

```
StringBuilder gettysburg = new StringBuilder();
gettysburg.append(4);
gettysburg.append(" score and ");
gettysburg.append(7.0);
gettysburg.append(" years ago...");
System.out.println("Text: " + gettysburg);
```

```
Text: 4 score and 7.0 years ago...
```

Reassigning/Reinitializing StringBuilders

- StringBuilders can not be initialized or assigned using the assignment operator like Strings can.

```
String tree = "Oak";
tree = "Maple"; //OK
```

```
StringBuilder fish = new StringBuilder("Goldfish");
fish = "Tuna"; //Error
fish = new StringBuilder("Tuna"); //OK
```

Reassigning/Reinitializing StringBuilders

- Alternatively, you can set the StringBuilder's length to 0.
  - This will clear the existing character sequence of the StringBuilder.

```
StringBuilder fish = new StringBuilder("Goldfish");
fish.setLength(0);
fish.append("Tuna");
```

Inserting values into a StringBuilder

- First parameter of the insert method is the index to insert at; Second parameter is the value to insert.
  - Will be inserted immediately after the index.

```
StringBuilder sb = new StringBuilder();
sb.append("Atlantic Community College");

sb.insert(9, "Cape ");
System.out.println("Text: " + sb);
```

```
Text: Atlantic Cape Community College
```

Replacing portions of a StringBuilder

- The replace method allows you to replace a portion of the StringBuilder with a new value.
  - First parameter is the starting index (inclusive).
  - Second parameter is the ending index (not inclusive).
  - Third parameter is the string to replace that substring with.

```
StringBuilder sb = new StringBuilder("Today is Tuesday");
sb.replace(9, 11, "Wedn");
System.out.println("Text: " + sb);
```

```
Text: Today is Wednesday
```

Deleting portions of a StringBuilder

- The delete method allows you to remove a portion of the StringBuilder.
  - First parameter is the starting index (inclusive).
  - Second parameter is the ending index (not inclusive).

```
StringBuilder sb = new StringBuilder("ABCDE");
sb.delete(2, 4);
System.out.println("Text: " + sb);
```

```
Text: ABE
```

Converting a StringBuilder to a String

- The toString method returns the value of the StringBuilder in String form.

```
StringBuilder sb = new StringBuilder("Atlantic City");

String copy1 = sb; //Compile Error
String copy2 = sb.toString(); //OK
```

Tokenizing Strings

- ***Tokenization*** is the process of splitting up a String into smaller units.
  - Strings are tokenized using a ***delimiter*** (Usually spaces or commas but can be anything.)
  - For example, the String “I heart New York” could be tokenized using whitespace as the delimiter which would break it up into 4 separate Strings: “I” “heart” “New” and “York”.
  - *Comma Separated Values* (CSV) is a widely used and recognized format.
    - Each line in a file has values separated by commas.
    - Commonly used by spreadsheet and database programs for exporting and importing data across different applications.

StringTokenizer Class

- An object provided by Java for tokenizing Strings.
- Must be imported.

```
import java.util.StringTokenizer;
```

StringTokenizer Class

- Instantiates a StringTokenizer that uses whitespace (spaces, tabs, newlines) as the delimiter (its default functionality).

```
String stringToTokenize = "Mays Landing, NJ";

StringTokenizer tokens = new StringTokenizer(stringToTokenize);
```

- We can specify a delimiter and we will see how later in the lecture.

Getting the Number of Tokens in a StringTokenizer

- The StringTokenizer's countTokens method returns the number of tokens.
  - It's good for checking if the String contained the expected number of tokens to weed out bad lines.
    - Your program expects ten tokens in the String but only contained 9, or something like that.

```
String stringToTokenize = "Mays Landing, NJ";
StringTokenizer tokens = new StringTokenizer(stringToTokenize);
System.out.println("Total tokens= " + tokens.countTokens());
```

```
Total tokens= 3
```

Determining if there are unread Tokens in a StringTokenizer

- The StringTokenizer's hasMoreTokens method returns true if there are more tokens left to be read.
  - This is good to use as the control in a while loop, to loop through the tokens.
  - When you take one of the tokens out of the StringTokenizer, it is removed. Eventually, after reading every token, this method will return false.

```
String stringToTokenize = "Mays Landing, NJ";
StringTokenizer tokens = new StringTokenizer(stringToTokenize);
System.out.println("Total tokens= " + tokens.countTokens());

while(tokens.hasMoreTokens()) {

}
```

Getting the Next Token in a StringTokenizer

- The StringTokenizer's nextToken method returns the next token.
  - The StringTokenizer works like a queue.
  - This method will keep returning the next token in the queue.
    - Will return null if there are no more tokens.

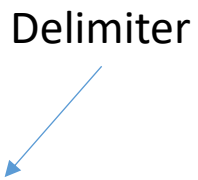
Getting the Next Token in a StringTokenizer

```
String stringToTokenize = "Mays Landing, NJ";
StringTokenizer tokens = new StringTokenizer(stringToTokenize);
System.out.println("Total tokens= " + tokens.countTokens());

while(tokens.hasMoreTokens()) {
  //Print Each Token
  System.out.println("Token: " + tokens.nextToken());

  //Print Remaining Tokens
  System.out.println("Tokens left: " + tokens.countTokens());
}

System.out.println("Done");
```

```
Total tokens= 3
Token: Mays
Tokens left: 2
Token: Landing,
Tokens left: 1
Token: NJ
Tokens left: 0
Done
```

StringTokenizer

- Instantiates a StringTokenizer that uses a supplied delimiter (can be any String).




```
String stringToTokenize = "Mays Landing, NJ";

StringTokenizer tokens = new StringTokenizer(stringToTokenize, ",");
```

StringTokenizer

```
String stringToTokenize = "Mays Landing, NJ";
StringTokenizer tokens = new StringTokenizer(stringToTokenize, ",");
System.out.println("Total tokens= " + tokens.countTokens());

while(tokens.hasMoreTokens()) {
  //Print Each Token
  System.out.println("Token: " + tokens.nextToken());

  //Print Remaining Tokens
  System.out.println("Tokens left: " + tokens.countTokens());
}

System.out.println("Done");
```

```
Total tokens= 2
Token: Mays Landing
Tokens left: 1
Token:  NJ
Tokens left: 0
Done
```

StringTokenizer (multiple delimiters)

The characters are treated individually

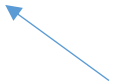
```
String stringToTokenize = "W. Atlantic City, NJ";
StringTokenizer tokens = new StringTokenizer(stringToTokenize, ",.");
System.out.println("Total tokens= " + tokens.countTokens());

while(tokens.hasMoreTokens()) {
  //Print Each Token
  System.out.println("Token: " + tokens.nextToken());

  //Print Remaining Tokens
  System.out.println("Tokens left: " + tokens.countTokens());
}

System.out.println("Done");
```

```
Total tokens= 3
Token: W
Tokens left: 2
Token:  Atlantic City
Tokens left: 1
Token:  NJ
Tokens left: 0
Done
```

String's split method

- Strings have a method (split) that can tokenize a String into an array.

```
String stringToTokenize = "Alabama Alaska Arkansas Arizona";

String[] tokens = stringToTokenize.split(" ");
```



Delimiter

Tip: The following will use any amount of whitespace as the delimiter:

```
String[] tokens = stringToTokenize.split("\\s+");
```

String's split method

```
String stringToTokenize = "Mays Landing, NJ";
String[] tokens = stringToTokenize.split(" ");

System.out.println("Total tokens= " + tokens.length);

for(String token : tokens) {
  //Print Each Token
  System.out.println("Token: " + token);
}

System.out.println("Done");
```

```
Total tokens= 3
Token: Mays
Token: Landing,
Token: NJ
Done
```

String's split method

```
String stringToTokenize = "Mays Landing, NJ";
String[] tokens = stringToTokenize.split(",");

System.out.println("Total tokens= " + tokens.length);

for(String token : tokens) {
  //Print Each Token
  System.out.println("Token: " + token);
}

System.out.println("Done");
```

```
Total tokens= 2
Token: Mays Landing
Token:  NJ
Done
```

String's split method (multiple delimiters)

```
String stringToTokenize = "W. Atlantic City, NJ";
String[] tokens = stringToTokenize.split("[,.]");

System.out.println("Total tokens= " + tokens.length);

for(String token : tokens) {
  //Print Each Token
  System.out.println("Token: " + token);
}

System.out.println("Done");
```

```
Total tokens= 3
Token: W
Token:  Atlantic City
Token:  NJ
Done
```

String's split method (multiple delimiters)

```
String stringToTokenize = "A.C.,NJ";
String[] tokens = stringToTokenize.split("[,.]");

System.out.println("Total tokens= " + tokens.length);

for(String token : tokens) {
  //Print Each Token
  System.out.println("Token: " + token);
}

System.out.println("Done");
```

```
Total tokens= 10
Token: A
Token: C
Token:
Token: NJ
```

This is because we had an instance where a . was right next to an ,

String's trim method

- The trim method removes leading and trailing whitespace.

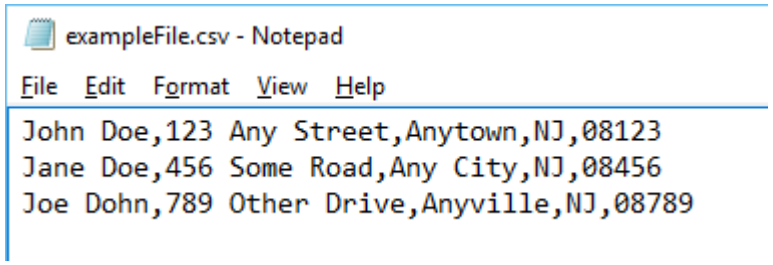
```
String stringToTokenize = "   Alabama,Alaska,Arkansas,Arizona   ";
stringToTokenize = stringToTokenize.trim();
String[] tokens = stringToTokenize.split(",");

for(String token : tokens) {
  //Print Each Token
  System.out.println("Token: " + token);
}

System.out.println("Done");
```

```
Token: Alabama.
Token: Alaska.
Token: Arkansas.
Token: Arizona.
Done
```

CSV files

- ***Comma separated values*** (or CSV) is a widely recognized text file format where each line of the file contains values that are separated by commas.

- Many database and spreadsheet programs use CSV format to export and import data.



Reading CSV files

- There is no special object for reading a CSV file.
  - Read the file as you would read any text file.
  - For each line in the file, split the line using a comma as the delimiter.

```
File myTextFile = new File("C:\\path\\to\\my\\exampleFile.csv");
Scanner fileReader = new Scanner(myTextFile);

while(fileReader.hasNextLine() == true) {
    String[] values = fileReader.nextLine().split(",");
    //The values array contains all of the values in the line.
}

fileReader.close();
```

Writing CSV files

- There is no special object for writing a CSV file.
  - Write the comma separated values as you would normally write to a file.

```
File myCsvFile = new File("C:\\path\\to\\my\\exampleFile2.csv");
PrintWriter fileWriter = new PrintWriter(myCsvFile);
fileWriter.println("Value1,Value2,Value3");
fileWriter.println("Value1,Value2,Value3");
fileWriter.println("Value1,Value2,Value3");
fileWriter.close();
```