AP Computer Science A

Java Programming Essentials [Ver. 2.0]

Unit 1: Elementary Programming

WEEK 5: CHAPTER 3- BASIC JAVA API [PART 2 STRING]

DR. ERIC CHOU IEEE SENIOR MEMBER

Objectives

- String is the most powerful class for text processing in Java.
- String Class Part 1: Basic Methods
- String Class Part 2: String Read-In and Comparison
- String Class Part 3: Substring
- Special String as Character Sets
- Formatted Print: **printf**

Overview

LECTURE 1

String

- The String class represents character strings. All string literals in Java programs, such as "abc", are implemented as instances of this class.
- Strings are constant; their values cannot be changed after they are created. String buffers support mutable strings. Because String objects are immutable they can be shared. For example:

```
String str = "abc";
```

is equivalent to:

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

String

- Here are some more examples of how strings can be used:

```
System.out.println("abc");
String cde = "cde";
System.out.println("abc" + cde);
String c = "abc".substring(2,3);
String d = cde.substring(1, 2);
```

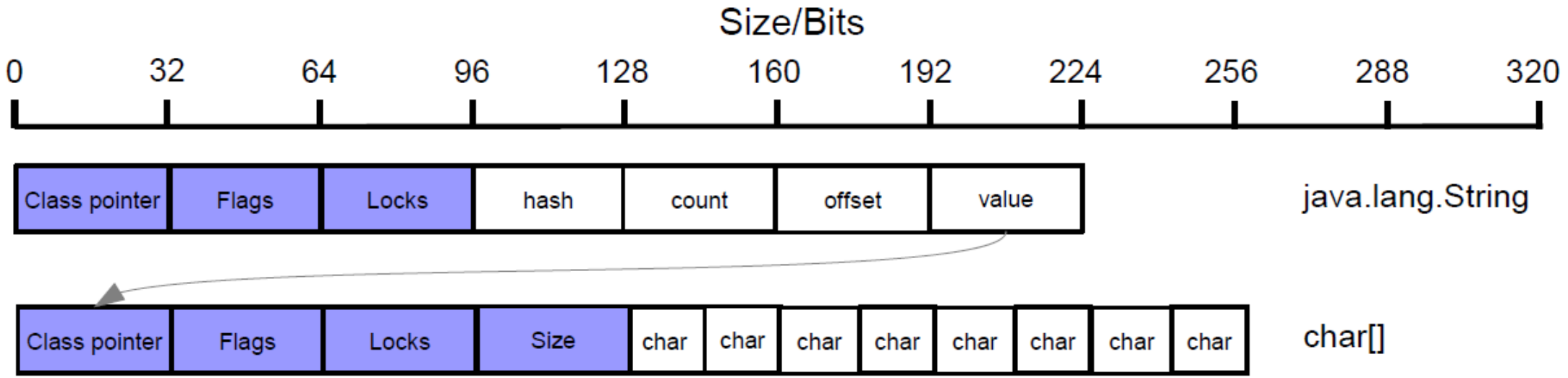
String Methods

- The class String includes methods for examining individual characters of the sequence, for comparing strings, for searching strings, for extracting substrings, and for creating a copy of a string with all characters translated to uppercase or to lowercase. Case mapping is based on the Unicode Standard version specified by the Character class.
- The Java language provides special support for the string concatenation operator (+), and for conversion of other objects to strings. For additional information on string concatenation and conversion, see The Java™ Language Specification.

String

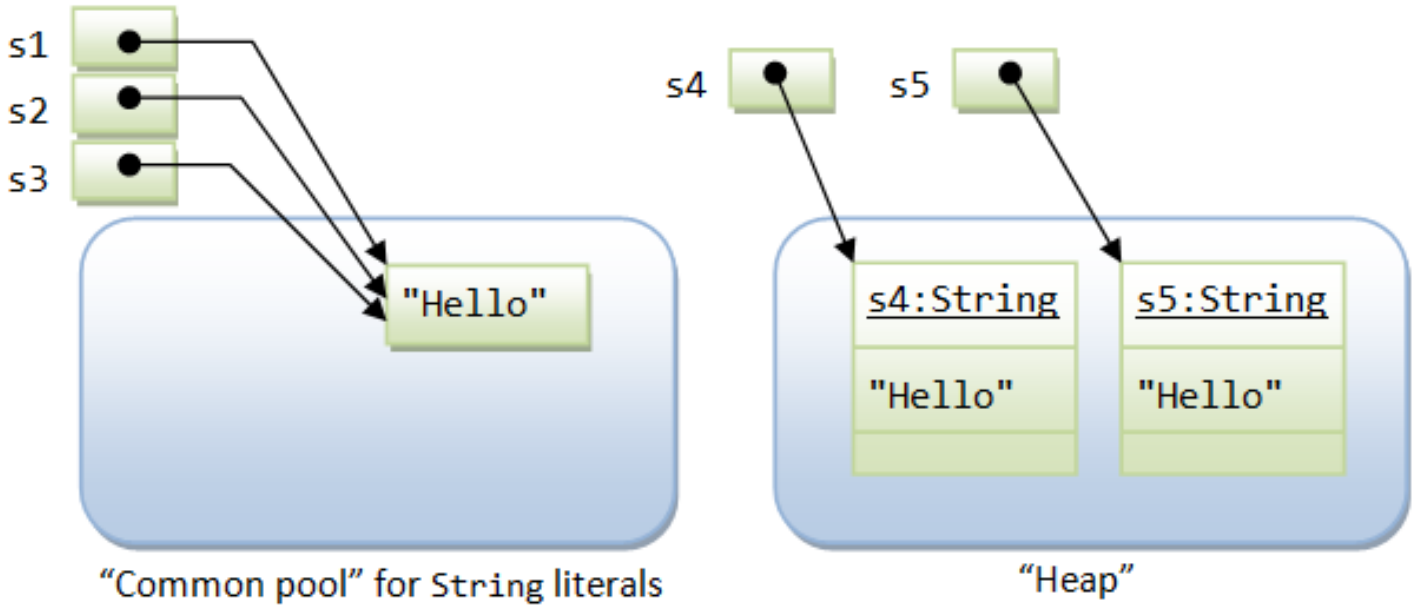
- A **String** represents a string in the UTF-16 format in which supplementary characters are represented by surrogate pairs (see the section Unicode Character Representations in the Character class for more information). Index values refer to char code units, so a supplementary character uses two positions in a String.
- The **String** class provides methods for dealing with Unicode code points (i.e., characters), in addition to those for dealing with Unicode code units (i.e., char values).
- Unless otherwise noted, methods for comparing Strings do not take locale into account. The Collator class provides methods for finer-grain, locale-sensitive **String** comparison.

- Simple example: java.lang.String containing “MyString”:



- 128 bits of char data, stored in 480 bits of memory, size ratio of 3.75 : 1
  - Maximum overhead would be 24:1 for a single character!

String Literal vs. String Object



String s1 = "Hello"; // String literal

String s2 = "Hello"; // String literal

String s3 = s1; // same reference

String s4 = new String("Hello"); // String object

String s5 = new String("Hello"); // String object

String Class

Part 1: Basic Methods

LECTURE 2

String Class

- Often you encounter the problems that involve string processing and file input and output. Suppose you need to write a program to replace all occurrences of a word with a new word in a file. How do you solve this problem?
- This chapter introduces strings and text files, which will enable you to solve this problem.

The `String` Class

- Constructing a String:
  - `String message = "Welcome to Java";`
  - `String message = new String("Welcome to Java");`
  - `String s = new String();`
- Obtaining String length and Retrieving Individual Characters in a string
- String Concatenation (concat)
- String Conversions

The String Type

A string is a sequence of characters.

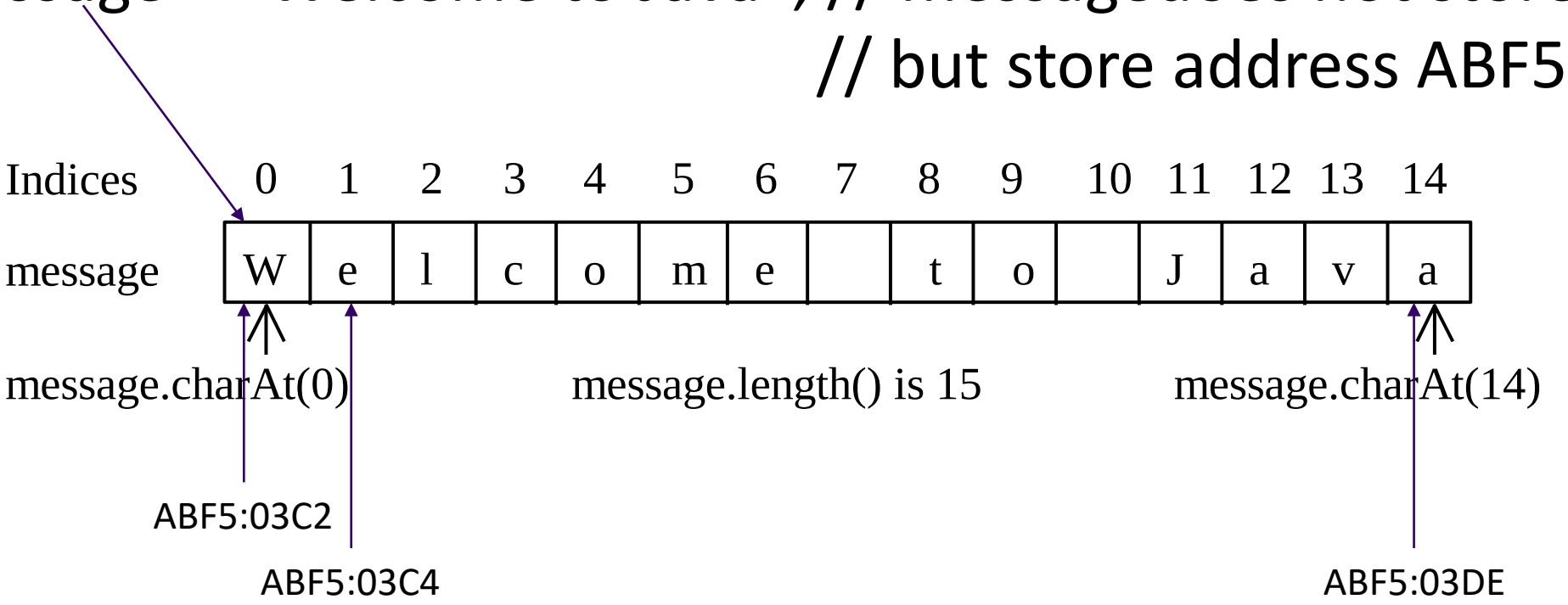
The char type represents only one character. To represent a string of characters, use the data type called **String**.

String is a predefined class in the Java library, just like the classes **System** and **Scanner**. The **String** type is not a primitive type. It is known as a ***reference type***.

Method	Description
length()	Returns the number of characters in this string.
charAt(index)	Returns the character at the specified index from this string.
concat(s1)	Returns a new string that concatenates this string with string s1.
toUpperCase()	Returns a new string with all letters in uppercase.
toLowerCase()	Returns a new string with all letters in lowercase
trim()	Returns a new string with whitespace characters trimmed on both sides.

Reference Data Type (only a pointer)

String message = “Welcome to Java”; // messagedoes not store the data
// but store address ABF5:03C2



ABF5:03C2 + **0x1C(28d=14d*2)**

Strings Are Immutable

A String object is immutable; its contents cannot be changed. Does the following code change the contents of the string?

```
String s = "Java";

s = "HTML";
```

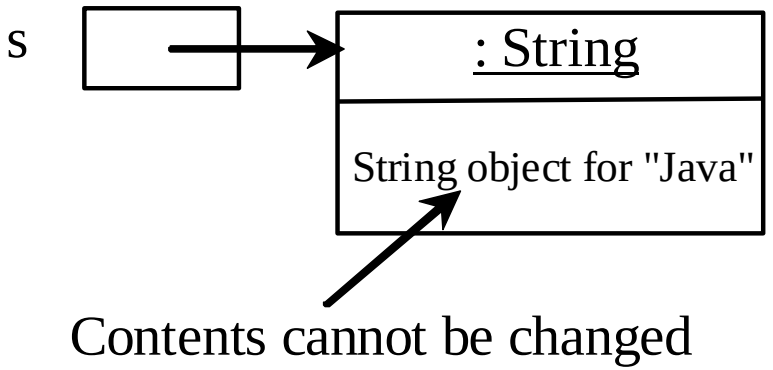
- s is only a reference (pointer).

Trace Code

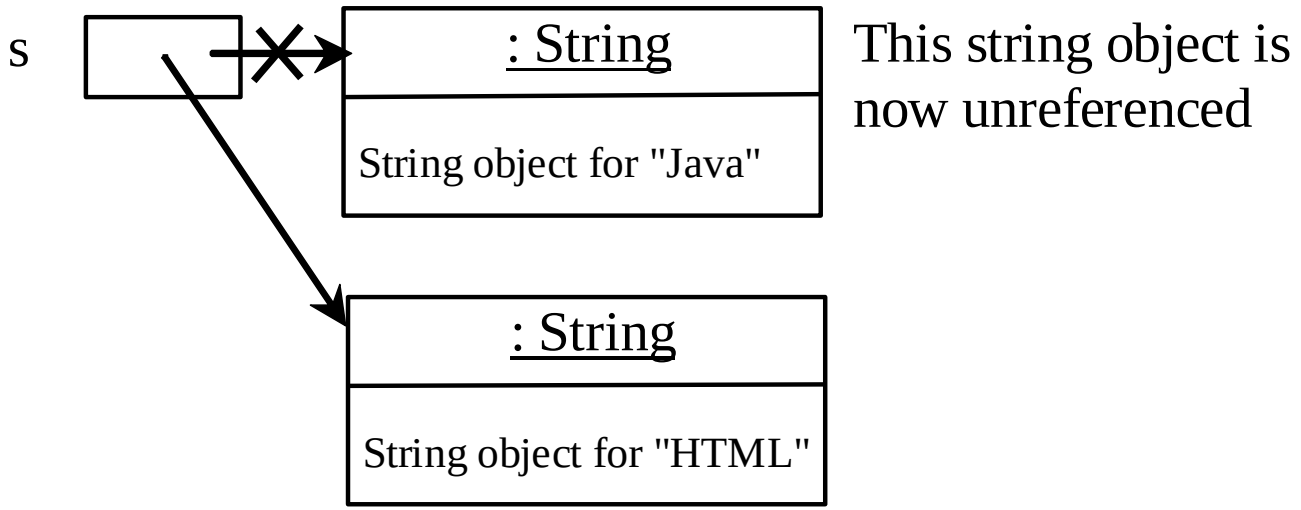
String s = "Java";

s = "HTML";

After executing `String s = "Java";`



After executing `s = "HTML";`



Interned Strings

- Since strings are immutable and are frequently used, to improve efficiency and save memory, the JVM uses a unique instance for string literals with the same character sequence.
- Such an instance is called **interned**. For example, the following statements: (see next page)

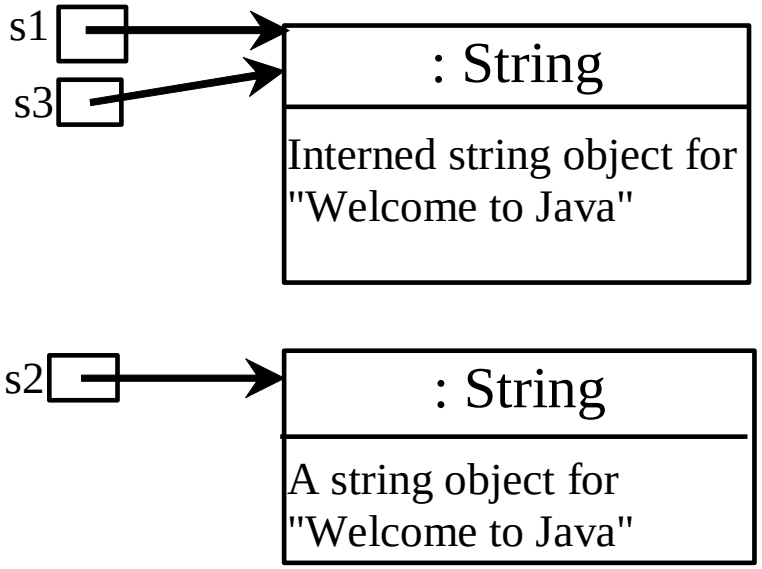
Examples

```
String s1 = "Welcome to Java";

String s2 = new String("Welcome to Java");

String s3 = "Welcome to Java";

System.out.println("s1 == s2 is " + (s1 == s2));
System.out.println("s1 == s3 is " + (s1 == s3));
```



Examples

display

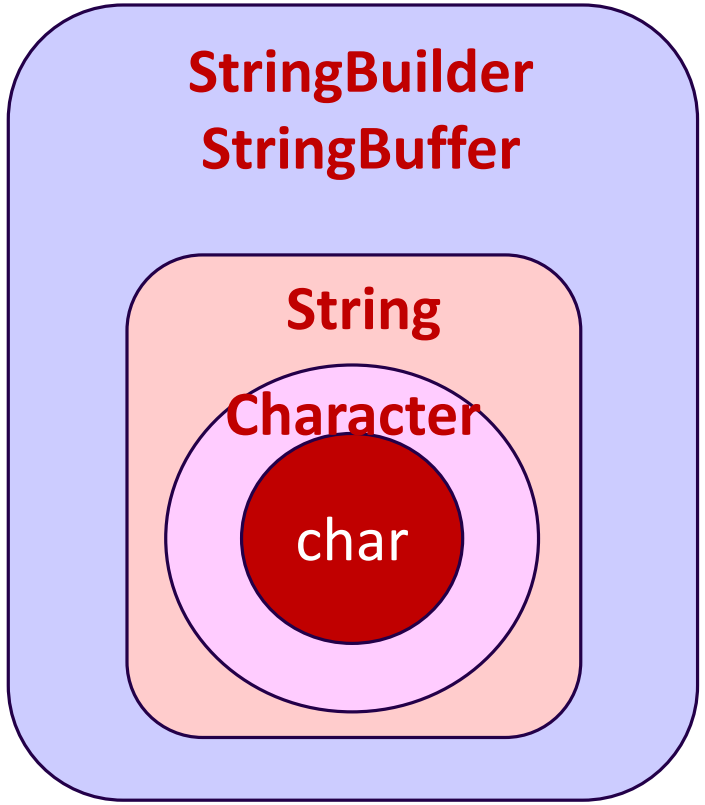
s1 == s2 is false

s1 == s3 is true

A new object is created if you use the new operator.

If you use the string initializer, no new object is created if the interned object is already created.

String Class Overview



char is primitive data type.
Character is char's wrapper class.
String is a collection of immutable **char**s.
StringBuilder is mutable version of **String**.
StringBuffer is synchronized version of **StringBuilder**.

String Class

Part 2: String Read-In and Comparison

LECTURE 3

Concatenating Strings

1. By concat() function,

   String s3 = s1.concat(s2);

2. By addition,

   String s3 = s1 + s2;

3. By increment assignment,

   message += “and Java is fun”;

String Concatenation

String s3 = s1.concat(s2);

String s3 = s1 + s2;

s1 + s2 + s3 + s4 + s5 same as

(((s1.concat(s2)).concat(s3)).concat(s4)).concat(s5);

Converting Strings

Lowercase:

“Welcome”.toLowerCase() -> “welcome”

Uppercase:

“Welcome”.toUpperCase() -> “WELCOME”

Trim: get rid of ‘ ‘, \t (tab), \f (new page), \r (carriage retrun), \n (new line)(whitespace letters)

“\t Good Night \n”.trim() -> “Good Night”

Reading Strings from Console

1. Step up input Stream:

```
Scanner input = new Scanner(System.in);  // input is a scanner (for scan codes)
                                         // System.in is a input stream
```

2. String s1 = input.next() // get the next string (before whitespace letters)

3. String s2 = input.nextLine() //get the next string before next return key (enter key)

4. do not use nextLine after nextByte(), nextShort(), nextInt(), nextFloat(0, nextDouble(), or next() // feed through of “\n” new line mark.

Reading next character from Console

```
System input = new Scanner(System.in);
System.out.print(”Enter a character: ”);
String s = input.nextLine();
char ch = s.charAt(0);
System.out.println(“The character entered is ” + ch);
```

Conversion between Strings and Numbers

Convert a string into a integer:

int intValue = Integer.parseInt(intString);

Convert a string into a double value:

double doubleValue = Double.parseDouble(doubleString);

Integer and Double classes are both included in **java.lang** package, and thus they are automatically imported.

Convert a number (char, boolean) into a string:

String s = number + ””;

Convert Character and Numbers to Strings

- The String class provides several static **valueOf** methods for converting a character, an array of characters, and numeric values to strings.
- These methods have the same name **valueOf** with different argument types char, char[], double, long, int, and float.
- For example, to convert a double value to a string, use **Double.valueOf(5.44)**. The return value is string consists of characters ‘5’, ‘.’, ‘4’, and ‘4’.

Convert the Read-in String to int, double

Convert Console Input String to integer:

```
int Value = Integer.parseInt(input.nextLine());
int Value = Integer.parseInt(input.next());
```

Convert Console Input String to integer:

```
double Value = Double.parseDouble(input.nextLine());
double Value = Double.parseDouble(input.next());
```

Comparing Strings

Method	Description
equals(s1)	Returns true if this string is equal to string s1.
equalsIgnoreCase(s1)	Returns true if this string is equal to string s1; it is case insensitive.
compareTo(s1)	Returns an integer greater than 0, equal to 0, or less than 0 to indicate whether this string is greater than equal to or less than s1.
compareToIgnoreCase(s1)	Same as compareTo except that the comparison is case insensitive.
startsWith(prefix)	Returns true if this string starts with the specified prefix.
endsWith(suffix)	Returns true if this string ends with the specified suffix.
contains(s1)	Returns if s1 is a substring in this string.

java.lang.String	
+equals(s1: Object): boolean	Returns true if this string is equal to string s1.
+equalsIgnoreCase(s1: String): boolean	Returns true if this string is equal to string s1 case-insensitive.
+compareTo(s1: String): int	Returns an integer greater than 0, equal to 0, or less than 0 to indicate whether this string is greater than, equal to, or less than s1.
+compareToIgnoreCase(s1: String): int	Same as compareTo except that the comparison is case-insensitive.
+regionMatches(toffset: int, s1: String, offset: int, len: int): boolean	Returns true if the specified subregion of this string exactly matches the specified subregion in string s1.
+regionMatches(ignoreCase: boolean, toffset: int, s1: String, offset: int, len: int): boolean	Same as the preceding method except that you can specify whether the match is case-sensitive.
+startsWith(prefix: String): boolean	Returns true if this string starts with the specified prefix.
+endsWith(suffix: String): boolean	Returns true if this string ends with the specified suffix.

String Comparisons

equals(Object object):

```
String s1 = new String("Welcome“);
String s2 = "welcome";
  if (s1.equals(s2)){  // false
    // if s1 and s2 have the same contents
  }
  if (s1 == s2) { // false
    // if s1 and s2 have the same reference
  }
```

String Comparisons, cont.

compareTo(Object object):

```
String s1 = new String("Welcome“);
String s2 = "welcome";
  if (s1.compareTo(s2) > 0) {
    // s1 is greater than s2
  }
  else if (s1.compareTo(s2) == 0) {
    // s1 and s2 have the same contents
  }
  else {
     // s1 is less than s2
  }
```

Demonstration Program

STUDENTINFO.JAVA

String Class

Part 3: Substring

LECTURE 4

Extracting Substrings

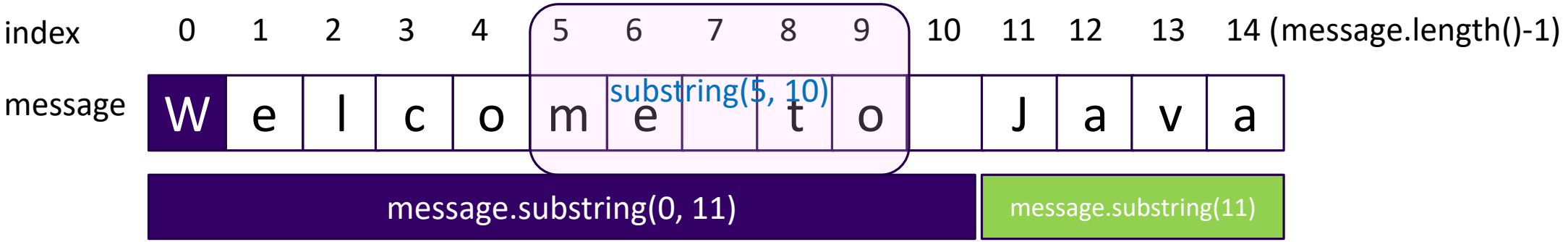
java.lang.String
+substring(beginIndex: int): String +substring(beginIndex: int, endIndex: int): String

Returns this string's substring that begins with the character at the specified beginIndex and extends to the end of the string, as shown in Figure 8.6.

Returns this string's substring that begins at the specified beginIndex and extends to the character at index endIndex – 1, as shown in Figure 8.6. Note that the character at endIndex is not part of the substring.

Obtaining Substrings

Methods	Description
substring(beginIndex)	Returns this string's substring that begins with the character at the specified beginIndex and extends to the end of the string.
substring(beginIndex, endIndex)	Returns this string's substring that begins at the specified beginIndex and extends to the character at index endIndex-1.



Finding a Character or a Substring in a String

java.lang.String	
+indexOf(ch: char): int	Returns the index of the first occurrence of ch in the string. Returns -1 if not matched.
+indexOf(ch: char, fromIndex: int): int	Returns the index of the first occurrence of ch after fromIndex in the string. Returns -1 if not matched.
+indexOf(s: String): int	Returns the index of the first occurrence of string s in this string. Returns -1 if not matched.
+indexOf(s: String, fromIndex: int): int	Returns the index of the first occurrence of string s in this string after fromIndex. Returns -1 if not matched.
+lastIndexOf(ch: int): int	Returns the index of the last occurrence of ch in the string. Returns -1 if not matched.
+lastIndexOf(ch: int, fromIndex: int): int	Returns the index of the last occurrence of ch before fromIndex in this string. Returns -1 if not matched.
+lastIndexOf(s: String): int	Returns the index of the last occurrence of string s. Returns -1 if not matched.
+lastIndexOf(s: String, fromIndex: int): int	Returns the index of the last occurrence of string s before fromIndex. Returns -1 if not matched.

Finding a Character or a Substring in a String

Method	Description
indexOf(ch)	Returns the index of the first occurrence of ch in the string. Returns -1 if not matched.
indexOf(ch, fromIndex)	Returns the index of the first occurrence of ch after fromIndex in the string. Returns -1 if not matched.
indexOf(s)	Returns the first occurrence of string s in this string. Returns -1 if not matched
indexOf(s, fromIndex)	Returns the index of the first occurrence of string s in this string after fromIndex. Returns -1 if not matched.
lastIndexOf(ch)	Returns the index of the last occurrence of ch in the string . Returns -1 if not matched.
lastIndexOf(ch,fromIndex)	Returns the index of the last occurrence of ch before fromIndex in this string. Returns -1 if not matched.
lastIndexOf(s)	Returns the index of the last occurrence of string s. Returns -1 if not matched.
lastIndexOf(s, fromIndex)	Returns the index of the last occurrence of string s **before** fromIndex -1 if not matched.

Finding a Character or a Substring in a String

`"Welcome to Java".indexOf('W')`	**returns 0.**
`"Welcome to Java".indexOf('x')`	**returns -1 .**
`"Welcome to Java".indexOf('o', 5)`	**returns 9.**
`"Welcome to Java".indexOf("come")`	**returns 3.**
`"Welcome to Java".indexOf("Java", 5)`	**returns 11.**
`"Welcome to Java".indexOf("java", 5)`	**returns -1.**
`"Welcome to Java".lastIndexOf('a')`	**returns 14.**

Converting, Replacing and Splitting

java.lang.String	
+toLowerCase(): String	Returns a new string with all characters converted to lowercase.
+toUpperCase(): String	Returns a new string with all characters converted to uppercase.
+trim(): String	Returns a new string with blank characters trimmed on both sides.
+replace(oldChar: char, newChar: char): String	Returns a new string that replaces all matching character in this string with the new character.
+replaceFirst(oldString: String, newString: String): String	Returns a new string that replaces the first matching substring in this string with the new substring.
+replaceAll(oldString: String, newString: String): String	Returns a new string that replace all matching substrings in this string with the new substring.
+split(delimiter: String): String[]	Returns an array of strings consisting of the substrings split by the delimiter.

Examples

`"Welcome".toLowerCase()` **returns a new string, welcome.**

`"Welcome".toUpperCase()` **returns a new string, WELCOME.**

`"  Welcome  ".trim()` **returns a new string, Welcome.**

`"Welcome".replace('e', 'A')` **returns a new string, WAlcomA.**

`"Welcome".replaceFirst("e", "AB")` **returns a new string, WABlcome.**

`"Welcome".replace("e", "AB")` **returns a new string, WABlcomAB.**

`"Welcome".replace("el", "AB")` **returns a new string, WABcome.**

Splitting a String

```
String[] tokens =
"Java#HTML#Perl".split("#", 0);
for (int i = 0; i < tokens.length; i++)
  System.out.print(tokens[i] + " ");
```

displays

Java HTML Perl

Lab

SOCIAL SECURITY NUMBER

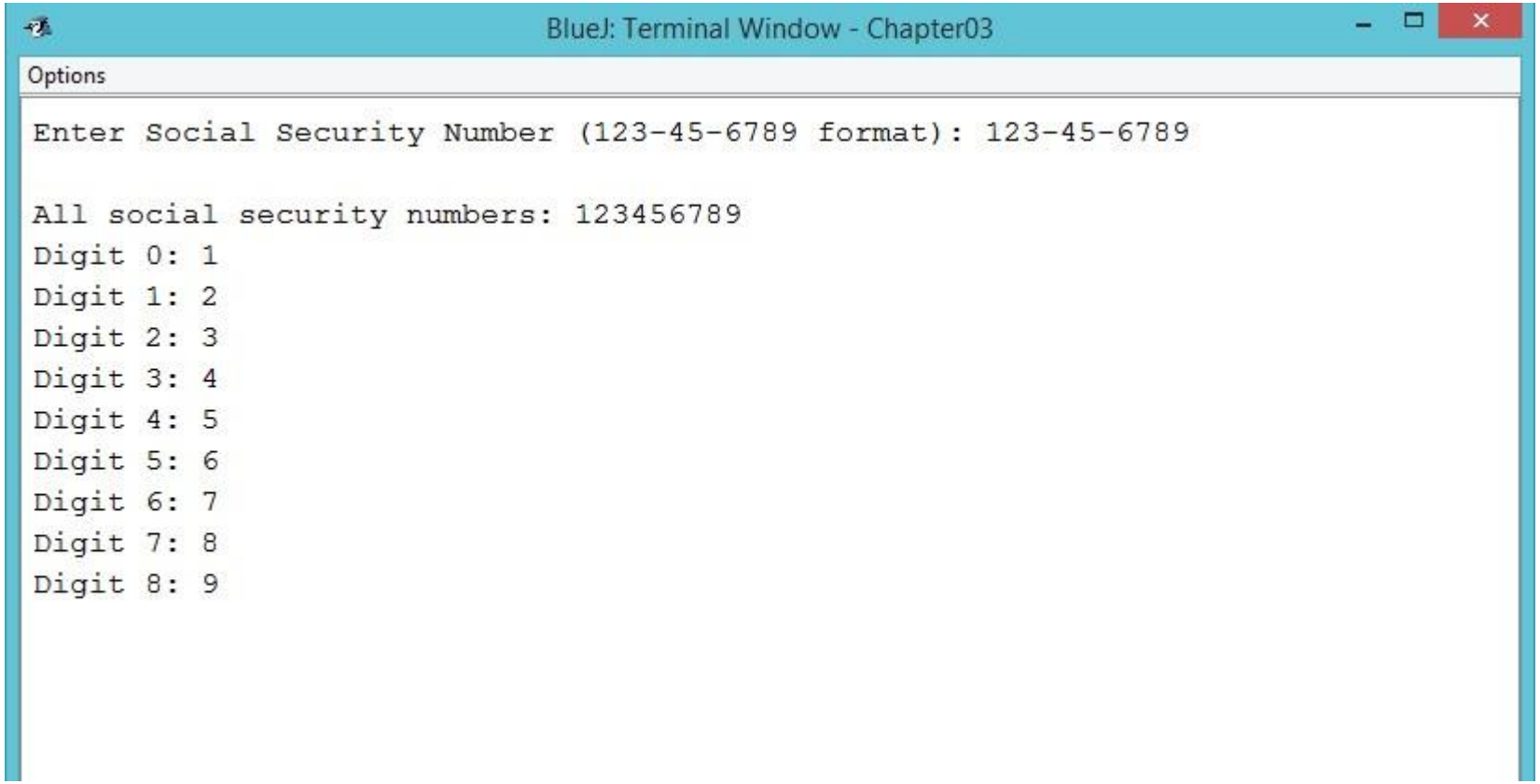
Lab: SSN.java

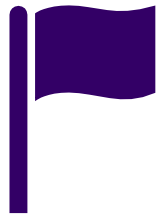
Read-in a social security number and print out proper information

- Write a program which read in a social security number in
- ###-##-#### format (# for a digit) or like 123-45-6789.
- Then, print out all digits in a line.
- After that, print out a digit in a line. Totally 9 lines.

Expected Result:

First un-guided lab. If you have issues, post your question in the discussion board or my facebook timeline.





Project: SSN.java

Student should work on this project in Class.

Lab

OMG + LOL

LAB: OMG + LOL

- Write a program to convert a message of 100 to 300 lower letters to a message of same meaning with common abbreviations. Try to replace at least five abbreviations (Intenet slangs).
- For example,

String originalMessage = “oh my god! i lost my high school backpack. you may laugh out loud but this is a very big deal. in case you didn’t know. just for your information.”;

LAB: OMG + LOL

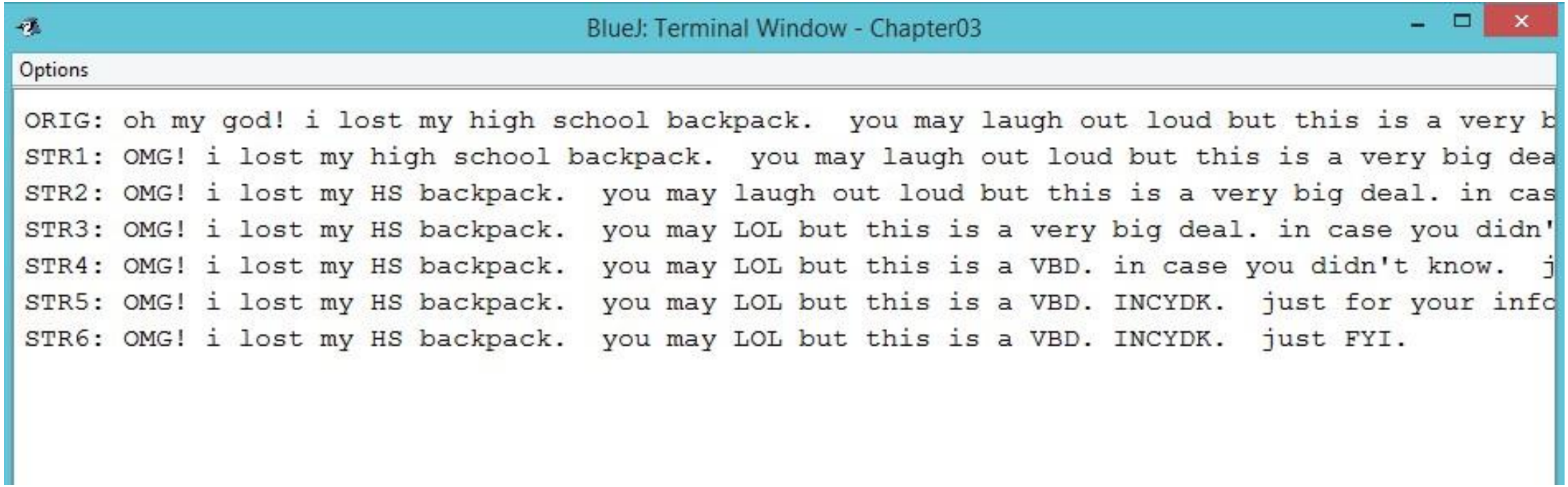
- Then, replace “oh my god!” to “OMG”, “high school” to “HS”, “laugh out loud” to “LOL”, “very big deal” to “VBD”, “in case you didn’t know” to “INCYDK”, and “for your information” to “FYI”.
- The result will be

  “OMG! i lost my HS backpack. you may LOL but this is a VBD. INCYDK. just FYI.”

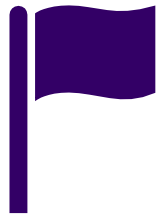
- **Check this web-site for some idea:** http://internetslang.com/
- **Hint:** use str.replace(), str.replaceAll();

Expected Result for the example:

You should try your own message and abbreviations.



The right edge was cut off because of the size of window and the length of message.



Project: ASCIIarts.java

Student should work on this project in Class.

Special String as Character Sets

LECTURE 5

name	R()	lgR()	characters
BINARY	2	1	01
DNA	4	2	ACTG
OCTAL	8	3	01234567
DECIMAL	10	4	0123456789
HEXADECIMAL	16	4	0123456789ABCDEF
PROTEIN	20	5	ACDEFGHIKLMNPQRSTVWY
LOWERCASE	26	5	abcdefghijklmnopqrstuvwxyz
UPPERCASE	26	5	ABCDEFGHIJKLMNOPQRSTUVWXYZ
BASE64	64	6	ABCDEFGHIJKLMNOPQRSTUVWXYZabcdef ghijklmnopqrstuvwxyz0123456789+/
ASCII	128	7	*ASCII characters*
EXTENDED_ASCII	256	8	*extended ASCII characters*
UNICODE16	65536	16	*Unicode characters*

Advantages

- Using these CharacterSet will help programming more efficient in many ways:
  1. Checking the substrings,
  2. Checking the set inclusion,
  3. Make finding traversal faster, and
  4. Create Histogram indexes.

Formatted Print: printf

LECTURE 6

Three print statements

print, println, printf

- **System.out.print(*argument*)** just prints out its argument, and
- **System.out.println(*argument*)** prints out its argument and ends the line.
- A third kind, **System.out.printf(*format*, *arguments*)**, gives more control over how things are printed.

Three print statements

print, println, printf

Example	Result
System.out.print("one"); System.out.print("two"); System.out.println("three");	onetwothree
System.out.println("one"); System.out.println("two"); System.out.println("three");	one two three
System.out.println();	[new line]

Print Statements

- A **print** "statement" is actually a call to the **print** or **println** method of the **System.out** object. The **print** method takes exactly one argument; the **println** method takes one argument or no arguments. However, the one argument may be a **String** which you create using the string concatenation operator **+**.
- If you ask to **print** an object, the **print** and **println** methods call that object's **toString()** method to get a printable string.

Example:

Example	Result

int x = 3;
int y = 5;
System.out.println(x + " and " + y + " is " + (x + y));
	3 and 5 is 8

int x = 3;
int y = 5;
System.out.println(x + " and " + y + " is " + x + y);
	3 and 5 is 35

Anything concatenated ("added") to a string is first converted to a string itself. In the second example above, the **x** is concatenated with the string, then the **y** is concatenated.

Escape Sequences for Special Characters

Escape Sequences			
Escape Sequence	Name	Unicode Code	Decimal Value
\b	Backspace	\u0008	8
\t	Tab	\u0009	9
\n	Linefeed	\u000A	10
\f	Formfeed	\u000C	12
\r	Carriage Return	\u000D	13
\\	Backslash	\u005C	92
\”	Double Quote	\u0022	34

\ is an escape character.

- The backslash is called an escape character. It is a special character. To display this character, you have to use an escape sequence ****. Escape character means the print function won't interpret the letters as it looks like. It will be printed in a special format.
- For example, the following code

```
System.out.println("\\t is a tab character");
```

- displays

\t is a tab character

- **The first \ is escape letter, the second \ (\\) will print \. It is trailed by t letter.**

Formatting Console Output

- **System.out.prinf:** You can use the System.out.printf method to display formatted output on the console.
- Formatted out provides programmer to output data in a specified data type and data precision.

Eg.

```
System.out.printf("Interest is $%4.2f",
                  12618.98*0.0013);
```

- Output is: Interest is $16.40

Format Specifier	Output	Example
%b	A Boolean value	True or false
%c	A character	‘a’
%d	A decimal integer	200
%f	A floating-point number	45.460000
%e	A number in standard scientific notation	4.556000e+01
%s	A string	“Java is cool”
%n	A newline mark	Like “\n” but better.
%tc	complete date and time	
%%	The character %	

Specifying Width and Precision

Example	Output
%5c	Output the character and add four spaces before the character item, because the width is 5.
%6b	Output the Boolean value and add one space before the false value and two spaces before the true value.
%5d	Output the integer item with width at least 5. If the number of digits in the item is < 5, add spaces before the number. If the number of digits in the item is > 5, the width is automatically increased.
%10.2f	Output the floating-point item with width at least 10 including a decimal point and two digits after the point. Thus, there are 7 digits allocated before the decimal point. If the number of digits before the decimal point in the item is < 7, add spaces > 7, the width is automatically increased.
%10.2e	Output the floating-point item with width at least 19 including a decimal point, two digits after the point and the exponent part. If the displayed number in scientific notation has width less than 10, add spaces before the number.
%12s	Output the string with width at least 12 characters. If the string item has fewer than 12 characters, add spaces before the string. If the string item has more than 12 characters, the width is automatically increased.

Formatted Print Out Example

LECTURE 7

Controlling integer width with printf

The %3d specifier means a minimum width of three spaces, which, by default, will be right-justified:

System.out.printf("%3d", 0);	0
System.out.printf("%3d", 123456789);	123456789
System.out.printf("%3d", -10);	-10
System.out.printf("%3d", -123456789);	-123456789

Left-justifying printf integer output

To left-justify integer output with printf, just add a minus sign (-) after the % symbol, like this:

System.out.printf("%-3d", 0);	0
System.out.printf("%-3d", 123456789);	123456789
System.out.printf("%-3d", -10);	-10
System.out.printf("%-3d", -123456789);	-123456789

The printf zero-fill option

To zero-fill your printf integer output, just add a zero (0) after the % symbol, like this:

System.out.printf("%03d", 0);	000
System.out.printf("%03d", 1);	001
System.out.printf("%03d", 123456789);	123456789
System.out.printf("%03d", -10);	-10
System.out.printf("%03d", -123456789);	-123456789

printf integer formatting

As a summary of printf integer formatting, here's a little collection of integer formatting examples. Several different options are shown, including a minimum width specification, left-justified, zero-filled, and also a plus sign for positive numbers.

Description	Code	Result
At least five wide	System.out.printf("'%5d'", 10);	' 10'
At least five-wide, left-justified	System.out.printf("'%-5d'", 10);	'10 '
At least five-wide, zero-filled	System.out.printf("'%05d'", 10);	'00010'
At least five-wide, with a plus sign	System.out.printf("'%+5d'", 10);	' +10'
Five-wide, plus sign, left-justified	System.out.printf("'%-+5d'", 10);	'+10 '

printf - floating point numbers

Here are several examples showing how to format floating-point numbers with printf:

Description	Code	Result
Print one position after the decimal	System.out.printf("'%.1f'", 10.3456);	'10.3'
Two positions after the decimal	System.out.printf("'%.2f'", 10.3456);	'10.35'
Eight-wide, two positions after the decimal	System.out.printf("'%8.2f'", 10.3456);	' 10.35'
Eight-wide, four positions after the decimal	System.out.printf("'%8.4f'", 10.3456);	' 10.3456'
Eight-wide, two positions after the decimal, zero-filled	System.out.printf("'%08.2f'", 10.3456);	'00010.35'
Eight-wide, two positions after the decimal, left-justified	System.out.printf("'%-8.2f'", 10.3456);	'10.35 '
Printing a much larger number with that same format	System.out.printf("'%-8.2f'", 101234567.3456);	'101234567.35'

printf string formatting

Here are several examples that show how to format string output with printf:

Description	Code	Result
A simple string	System.out.printf("'%s'", "Hello");	'Hello'
A string with a minimum length	System.out.printf("'%10s'", "Hello");	' Hello'
Minimum length, left-justified	System.out.printf("'%-10s'", "Hello");	'Hello '

Demonstration Program

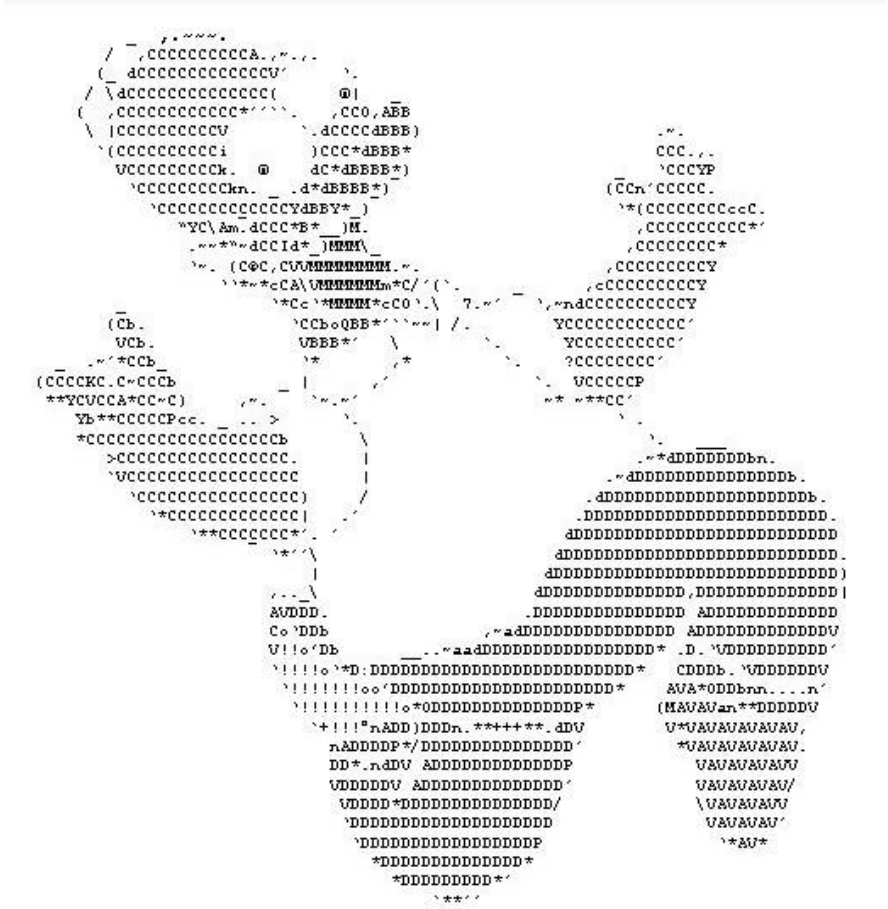
FORMATDEMO.JAVA

Lab

ASCII ARTS

ASCII Arts

from http://www.instructables.com/id/ASCII-Art/?ALLSTEPS



Refer some web-site like this one to create your own artwork.

Adjustment to fit into Java program after you have your artwork

Step 1: copy the artwork to notepad.

Step 2: replace the characters that need to be changed for escape sequence.

e.g.

**\ needs to be changed to **

“ needs to be changed to \”

Step 3:

add **System.out.println(“** and **“);** to wrap up the artwork.

Lab: ASCIIarts.java

- Write a program to print out an ASCII artwork, first create your own artwork or download from some web-site (e.g. **http://www.textfiles.com/art/**). Second, modify the ASCII artwork in notepad to Java statements. Finally, copy the statements into ASCIIarts.java file
- Compile and check the results.

Download

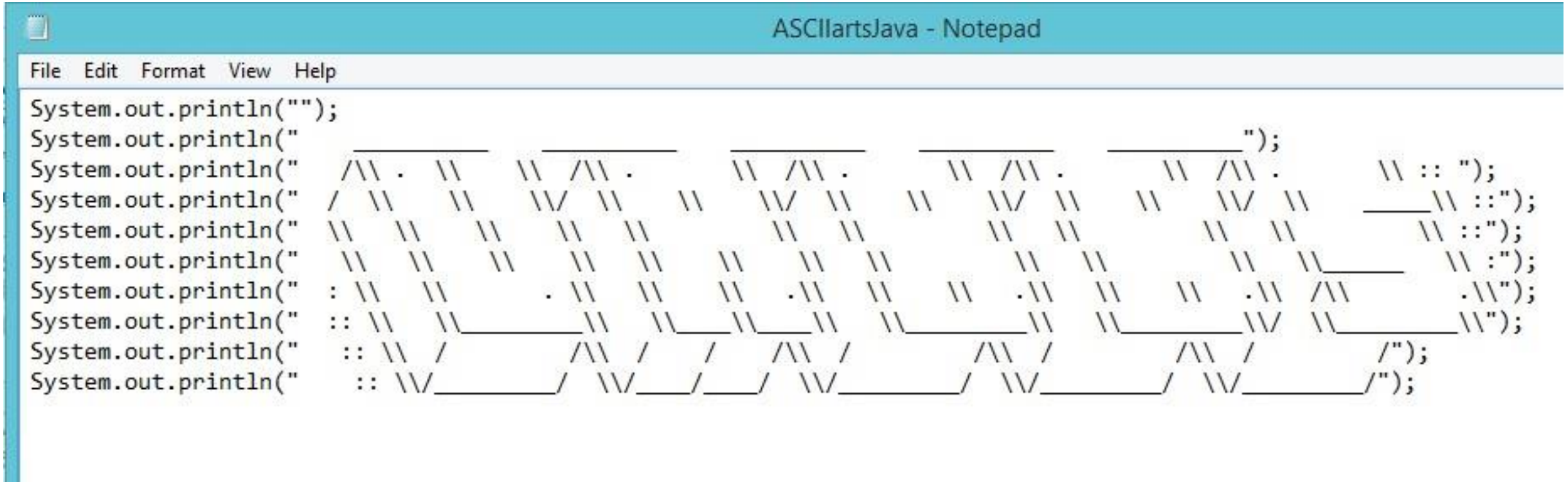
ASCIIarts.txt

ASCIIartsJava.txt

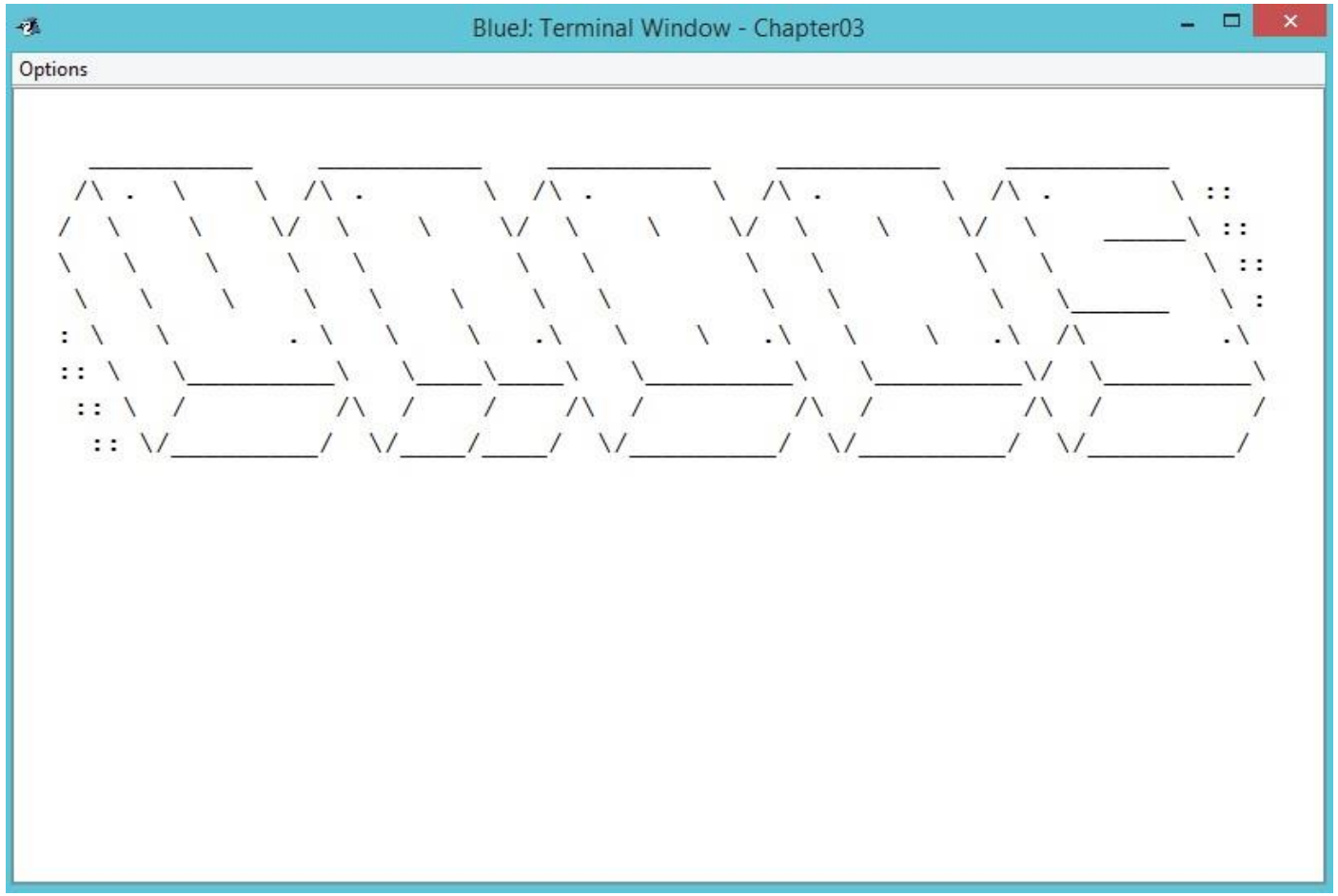
ASCIIarts.java

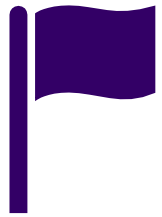
as a sample program project.

Example Source:
(in Java Statement format)



Expected Result for Sample Program:





Project: ASCIIarts.java

Student should work on this project in Class.

Chapter Project: Student GPA

LECTURE 8

Chapter Project: StudentGPA.java

- Write a program to read in student's information for one semester. The student information should include full name, social security number (SSN), date of birth (DOB), and address. Assume that this student took Math, English, Physics, Chemistry, P.E. and US History in this semester. He is awarded score in 0 to 5 format (all integers)
- Then, calculate for his GPA
- Then, print out the student's school semester score report. Print out a school header first.
- Then, student's information. Followed by Student's score subject by subject. Finally, provide
- his GPA in double format with proper format (2 significant digits).

BlueJ: Terminal Window - Chapter03

Options

```
Enter Student's Name (First, Last): John Smith

Enter Student's SSN (XXX-XX-XXXX): 123-45-6789

Enter Student's Day of Birth (MM/DD/YYYY): 01/01/1998

Enter Student's Address: 101 Adams Drive, Los Angeles, Ca 90007

Enter Math Score (0-5): 4

Enter English Score (0-5): 3

Enter Physics Score (0-5): 4

Enter Chemistry Score (0-5): 2

Enter P.E. Score (0-5): 1

Enter U.S. History Score (0-5): 5

                    Washington High School
                  Semester Score Report Card
Name: John Smith                      SSN: XXX-XX-6789  DOB: 01/01/1998
Address: 101 Adams Drive, Los Angeles, Ca 90007
=========================================================================
Math Score:       4  English Score:    3  Physics Score:    4
Chem. Score:      2  P.E. Score:       1  U.S.Hist. Score:  5
Student's GPA: 3.17
```

Expected Results:
StudentGPA.java (skeleton), StudentGPAAnswer.java (sample answer)

Assignment

STUDENT GPA

SUBMIT YOUR PROGRAM TO MOODLE COURSE UPLOAD LINK