Lecture 2

String

String

- Strings are the type of objects that can store the character of values. A string acts the same as an array of characters in Java. A String variable contains a collection of characters surrounded by double quotes.
- There are two ways to create a string in Java, **String Literal**, and using **new Keyword.**
- Syntax:
- String_Type string_variable = "sequence_of_string";

Ways of Creating a String

- String literal: String variable_name;
  - String firstname= "David";
  - String lastname= "Brown";
- Using new keyword:
  - String firstname= new String("David");

String method

- String has predefined functions contained within the class which we can use for our convenience to do string manipulations.
- **int length():** Returns the number of characters in the String.
- **char charAt(int i)**: Returns the character at ith index.
- **String substring (int i)**: Return the substring from the ith index character to end.
- **String substring (int i, int j):** Returns the substring from i to j-1 index.
- **String concat(String str):** Concatenates specified string to the end of this string.
- **int indexOf (String s):** Returns the index within the string of the first occurrence of the specified string.
- **int indexOf (String s, int i):** Returns the index within the string of the first occurrence of the specified string, starting at the specified index.
- **int lastIndexOf(String s):** Returns the index within the string of the last occurrence of the specified string.

String method

- **boolean equals(Object otherObj):** Compares this string to the specified object.
- **boolean equalsIgnoreCase (String anotherString):** Compares string to another string, ignoring case considerations.
- **int compareTo(String anotherString):** Compares two string lexicographically.
- **int compareToIgnoreCase(String anotherString):** Compares two string lexicographically, ignoring case considerations.
- **String toLowerCase():** Converts all the characters in the String to lower case.
- **String toUpperCase():** Converts all the characters in the String to upper case.
- **String trim():** Returns the copy of the String, by removing whitespaces at both ends. It does not affect whitespaces in the middle.
- **String replace (char oldChar, char newChar):** Returns new string by replacing all occurrences of *oldChar* with *newChar.*

Classes, Objects, and Methods

- A Java program works by having things called **objects** perform actions. The actions are known as **methods** and typically involve data contained in the object.

- All objects of the same kind are said to be of the same class. So, a **class** is a category of objects.

- When the object performs the action of a given method, it is called **invoking** the method (or **calling** the method). Information provided to the method in parentheses is called the **argument** (or arguments).

String length

- A String in Java is actually an object, which contain methods that can perform certain operations on strings.

- The length of a string can be found with the length() method.

```
String txt = “ABCDEFGHIJKLMNOPQRSTUVWXYZ";

System.out.println("The length of txt: " + txt.length());
```

Part of String

0	1	2	3	4	5	6	7	8	9	10
H	E	L	L	O		W	O	R	L	D

- charAt() function takes the index.
- System.out.println(txt.charAt(0)); // print ‘H’
- System.out.println(txt.charAt(1)); // print ‘E’
- System.out.println(txt.charAt(2)); // print ‘L’

String Methods

```
String txt = "Hello World”;

System.out.println(txt.toUpperCase());
// Outputs "HELLO WORLD”

System.out.println(txt.toLowerCase());
// Outputs "hello world”

// The indexOf() method returns the index (the position)
// of the first occurrence of a specified
// text in a string (including whitespace)

String txt = "Please locate where 'locate' occurs!”;

System.out.println(txt.indexOf("locate")); // Outputs 7
```

String Concatenation

- The + operator can be used between strings to combine them. This is called concatenation:

```
String s1=“David”;
String s2=“Brown”;
System.out.println(s1 + " " + s2);
// will print out “David Brown”
```

- The + operator can be used between strings to combine them.

```
string s1=“David”;
string s2=“Brown”;
System.out.println(s1.concat(s2));
// will print out “DavidBrown”
```

Adding Numbers and Strings

- Add a number and a string, the result will be a string concatenation

```
String x = “10";
int y = 20;
String z = x + y;
// z will be 1020 (a String)
```

Exercise

Exercise 1. Output a string one character at a time.

a) Request a text string of length five from the console.
b) Output the string one character at a time followed by a newline.

Example Output (input in bold italics)

```
Enter a text string: Hello
H
e
l
l
o
```

Exercise

```
Exercise 2. Output a string one word at a time.

a) Initialize a string variable with the text
“This is a text string.”
b) Use one or more string functions to isolate
and output each word.

Example Output (input in bold italics)
This
is
a
test
string
```

Operator

Java Operator

- Operators are used to perform operations on variables and values.
- Java divides the operators in the following groups:
  - Arithmetic operators (+, -, *, /, %, ++,- -)
  - Assignment operators (=, +=, -=, *=, /=, %=)
  - Comparison operators (>, >=, <, <=, ==, !=)
  - Logical operators (and, or, not)

Assignment Operator

Operator	Example	Same As
=	x = 5	x = 5
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3
%=	x %= 3	x = x % 3

Relational Operator

Operator	Example	Same As
==	Equal	x == y
!=	Not equal	x != y
>	Greater than	x > y
<	Less than	x < y
>=	Greater than or equal to	x >= y
<=	Less than or equal to	x <= y

Relational Operator

- **Relational operators** are used to check conditions like whether two values are equal, or whether one is greater than the other.

- The result of a relational operator is one of two special values, **true** or **false**. These values belong to the data type **boolean**; in fact, they are the only boolean values.

- The two sides of a relational operator have to be compatible. Most relational operators don’t work with strings. Instead, you should use the **equals** method

Logical Operator

Operator	Description	Example
and	Returns True if both statements are true	x < 5 && x < 10
or	Returns True if one of the statements is true	x < 5 | | x < 4
Not	Reverse the result, returns False if the result is true	! (x < 5 && x < 10)

Truth Table

A	B	A and B
T	T	T
T	F	F
F	T	F
F	F	F

A	B	A or B
T	T	T
T	F	T
F	T	T
F	F	F

A	not A
T	F
F	T

Logical Operator

- Java has three **logical operators**: &&, ||, and !, which respectively stand for *and*, *or*, and *not*. The results of these operators are similar to their meanings in English.
- Logical operators evaluate the second expression only when necessary.
- Negating a logical expression is the same as negating each term and changing the operator.

Exercise

Q3. Write a program that accepts two integers as input, divides the first by the second, and outputs the answer in floating point format. Example Program (input is **bolded** and italicized)

```
 Enter a numerator: 10
 Enter a denominator: 3
 Answer: 3.33
```

Exercise

Q4. Write a program that accepts a money value (i.e. $3.85) as input and then outputs the total number of quarters and remaining change. Change should be output in proper dollar format as displayed in the output example.
Example Program (input is **bolded** and italicized)

```
 Enter money: $3.42
 Number of quarters is 13
 Change is $0.17
```

Exercise

Q5. Write a program that requests two integers and outputs the answer in the form X remainder Y as follows: (Input in italics)

```
Simple Divider
--------------
Input first number: 5
Input second number: 3
Answer: 1 remainder 2
```

Branching

Conditional statements

- To write useful programs, we almost always need to check conditions and react accordingly. **Conditional statements** give us this ability. The simplest conditional statement in Java is the **if** statement.

- A second form of conditional statement has two possibilities, indicated by **if** and **else**. The possibilities are called **branches**, and the condition determines which one gets executed

The if Statement

- Use the **if** statement to specify a block of Java code to be executed if a condition is **true**

```
if (condition) {
   // block of code to be executed
   // if the condition is true
}
```

The else Statement

- Use the **else** statement to specify a block of code to be executed if the condition is **false**.

```
if (condition) {
   // block of code to be executed
   // if the condition is true
} else {
   // block of code to be executed
   // if the condition is false
}
```

The else if Statement

- Use the **else if** statement to specify a new condition if the first condition is **false**.

```
if (condition1) {
    // block of code to be executed
    // if condition1 is true
} else if (condition2) {
    // block of code to be executed
    // if the condition1 is false and condition2 is true
} else {
    // block of code to be executed
    // if the condition1 is false and condition2 is false
}
```

Short Hand If...Else

- There is also a short-hand if else, which is known as the **ternary operator** because it consists of three operands.

  Syntax:

  ```
  variable = (condition) ? expressionTrue :  expressionFalse;
  ```

  Example:

  ```
  int time = 20;
  String result = (time < 18) ? "Good day." : "Good evening.”;
  System.out.println(result);
  ```

Exercise

Q6. Write a program that accepts an integer x as input, and outputs a 1 if x is even, or a 0 if the x is odd.

Q7. Write a program that accepts an integer x as input and outputs a 1 if x is not a multiple of 5, 0 otherwise.

Q8. Write a program that accepts an integer x as input and outputs a 1 if x is a multiple of 3 and not a multiple of 7, 0 otherwise.

Q9. Write a program that accepts an integer x as input and outputs:
- “X is a multiple of 5” if x is evenly divisible by 5.
- “X is a multiple of 3” if x is evenly divisible by 3.
- Both statements, if both conditions are true.

Output Example:
```
Enter an integer x: 15
15 is a multiple of 5.
15 is a multiple of 3.
```

Exercise

Q10. Write a program that requests if the user will a) go to the movies or b) go to dinner. The following should be output:

- "You are going to the movies" if a is true
- "You are going to dinner" if b is true
- "You must do something" if a and b are both false
- "You cannot do both" if a and b are both true

This program should include two Boolean variables to store the values of both possibilities. Mutual exclusion must apply.

Output Example:

```
Do you wish to go to the movies (1/0)? 0
Do you wish to go to Dinner (1/0)? 1
You are going to dinner.
```

Exercise

```
Q11. Write a program that asks if it is raining.
If it is raining offer the user two options:
1) Watch TV
2) Do homework.
If is not raining offer the user two options:
1) Hit the beach.
2) Attend a museum opening.
Once the user selects an option, output the
activity.
```

Exercise

```
Q12. Write a program that requests an integer x from 1 to 100
inclusive.
Validate the input by determining if:
a) the input is an integer and
b) the input is in the specified range.
Output the validation status to the user.
hint: use casting to determine if a number is an integer

Output Example 1:
Input an integer between 0 and 100 inclusive: 54
54 is an integer between 0 and 100 inclusive.

Output Example 2:
Input an integer between 0 and 100 inclusive: 5.4
5.4 is not an integer.

Output Example 3:
Input an integer between 0 and 100 inclusive: 500
500 is not in range 0 to 100.
```

Exercise

Q13. Write a program that accepts three distinct input variables x, y and z.
Output the values of these variables from largest to smallest.

Output Example:

```
Sort three distinct integers from Max to Min:
Input x: 40
Input y: 60
Input z: 30
Max to min: 60 40 30
```

Switch Statements

- Use the **switch** statement to select one of many code blocks to be executed.

```
switch(expression) {
    case x:
        // code block
        break;
    case y:
        // code block
        break;
    default:
        // code block
}
```

break & default

- **break**
  - When Java reaches a **break** keyword, it breaks out of the switch block.
  - This will stop the execution of more code and case testing inside the block.
  - When a match is found, and the job is done, it's time for a break. There is no need for more testing.
  - A break can save a lot of execution time because it "ignores" the execution of all the rest of the code in the switch block.
- **default**
  - The **default** keyword specifies some code to run if there is no case match
  - The **default** statement is used as the last statement in a switch block, it does not need a break.

Exercise

Q14. Write a program that emulates a soft drink vending machine. Use if or switch to handle a menu of the following options: 1) Coke, 2) Sprite,3) Water, 4) Ice tea.

Output Example: (input is in **bold**)

```
Vending Machine
1) Coke
2) Sprite
3) Water
4) Ice tea
Select: 4
You selected an Ice tea.
```

Exercise

Q15. Write a program that implements the five work days of the week as an enumeration data type. Use a switch statement to list the five days in a menu. The user then selects which day to take off.

```
Output Example:
Days of the week:
0) Monday
1) Tuesday
2) Wednesday
3) Thursday
4) Friday
Select a day to take off: 3
You selected Thursday.
```

Exercise

Q16. Copy program 13 to a new program. Remove the enumeration. Rewrite the program so that the user menu is selected by letter.

Output Example:

```
Days of the week:
a) Monday
b) Tuesday
c) Wednesday
d) Thursday
e) Friday
Select a day to take off: d
You selected Thursday.
```

StringTokenizer Class

- The StringTokenizer class is used to recover the words in a multiword string. It is often used when reading input.

- One approach to reading keyboard input is to read an entire line of input into a variable of type **String** for example, with the method **nextLine** of the **Scanner** class and then to use the **StringTokenizer** class to decompose the string in the variable into words.

StringTokenizer

- The class StringTokenizer is in the standard Java package (library) java.util. To tell Java where to find the class StringTokenizer, any class or program that uses the class StringTokenizer must contain the following (or something similar) at the start of the file.

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

Methods in StringTokenizer

- **public StringTokenizer(String theString)**: Constructor for a tokenizer that will use whitespace characters as separators when finding tokens in theString.

- **public StringTokenizer(String theString, String delimiters)**: Constructor for a tokenizer that will use the characters in the string delimiters as separators when finding tokens in theString.

- **public boolean hasMoreTokens()**: Tests whether there are more tokens available from this tokenizer's string. When used in conjunction with nextToken, it returns true as long as nextToken has not yet returned all the tokens in the string; returns false otherwise.

Methods in StringTokenizer

- **public String nextToken()**: Returns the next token from this tokenizer's string. (Throws NoSuchElementException if there are no more tokens to return.)

- **public String nextToken(String delimiters)**: First changes the delimiter characters to those in the string delimiters. Then returns the next token from this tokenizer's string. After the invocation is completed, the delimiter characters are those in the string delimiters. (Throws NoSuchElementException if there are no more tokens to return. Throws NullPointer-Exception if delimiters is null.)

- **public int countTokens()**: Returns the number of tokens remaining to be returned by nextToken.