**Administrative**

**Today’s session**

Regular expressions

Algorithms and pseudo code

Conditions

Relational operators

Logical operators

Short-circuit evaluation

*if* statement

*switch* statement

**Session Topics**

**Regular expressions**

● A **regular expression** is a pattern used to match strings.

● A regular expression may be used to:

✓ Validate user input.

✓ Search text for patterns.

✓ Find and optionally replace strings.

● A regular expression is a string itself composed of one or more of the following symbols:

| Symbol | Purpose |
| --- | --- |
| . | Match any single character. Use \\. if actual character is needed. |
| \d | Match any single digit. |
| \D | Match any single character not a digit. |
| \w | Match any single letter, digit, or underscore. |
| \W | Match any single character not a letter, digit, or underscore. |
| \s | Match any single whitespace character. |
| \S | Match any single character not a whitespace character. |
| [ ]\* | Match zero or more characters within the brackets. |
| [ ]+ | Match one or more characters within the brackets. |
| [ ]? | Match zero or one character within the brackets. |
| ( ) | Group content within the parentheses. |
| ( )\* | Match zero or more times content within the parentheses. |
| ( )+ | Match one or more characters content within the parentheses. |
| ( )? | Match zero or one content within the parentheses. |
| | | Match either side of the vertical bar operator. This operator has the lowest precedence. |
| {n} | Match the previous symbol n times. |
| {n,} | Match the previous symbol at least n times. |
| {n,m} | Match the previous symbol n to m times. |

● The full list of regular expression symbols is available at [docs.oracle.com/javase/8/docs/api/java/util/regex/Pattern.html#sum](https://docs.oracle.com/javase/8/docs/api/java/util/regex/Pattern.html#sum).

● A useful regular expression tutorial is available at [www.regular-expressions.info/index.html](http://www.regular-expressions.info/index.html).

● Here are some String methods using a regular expression on a string object. See the complete list at [docs.oracle.com/javase/8/docs/api/java/lang/String.html](http://docs.oracle.com/javase/8/docs/api/java/lang/String.html).

| Method | Purpose |
| --- | --- |
| matches(<regex>) | Return true if the string contains any <regex> patterns. |
| replaceAll(<regex>,<new-string>) | Replace all <regex> patterns in string with <new-string> . |
| split(<regex>) | Return string array with string parsed with <regex> pattern. |

● See **Regular expressions** sample application on Blackboard.

● See **String parser** sample application on Blackboard.

**Algorithms and pseudo code**

● An **algorithm** is the logic to perform a task or solve a problem.

● **Pseudo code** is an English-like description of an algorithm.

**Conditions**

● A **condition** controls the flow of processing in an application.

● A condition is also known as a **Boolean expression**.

● A condition is true or false.

● A condition typically uses relational and logical operators.

● A condition is used in conditional (if, switch) and iterative (while, for) statements.

**Relational operators**

● A **relational operator** compares two expressions of the same data type.

● A relational operator is used to form a condition.

● There are six relational operators:

| Relational Operator | Description |
| --- | --- |
| == | equals\* |
| != | not equal to |
| < | less than |
| <= | less than or equal to |
| > | greater than |
| >= | greater than or equal to |

\* The **== operator** is used to compare two primitive data types (byte, short, int, long, float, double, boolean, or char). The **equals method** is used to compare two objects.

**Logical operators**

● Combine two simple conditions into a **compound condition**, or reverse the value of a condition.

● Each simple condition is true or false, and after applying the logical operator, the compound condition is true or false.

● There are three logical operators:

| Logical Operator | Description |
| --- | --- |
| && | ● Means “and”.  ● If *both* sides of && are **true**, compound condition is **true**. |
| || | ● Means “or”.  ● If *either* side of || is **true**, compound condition is **true**. |
| ! | ● Means “not”.  ● If condition after ! is **false**, compound condition is **true**.  ● If condition after ! is **true**, compound condition is **false**. |

● A **truth table** shows the result when conditions are connected with a logical operator.

● **&& (and) operator:** the conditions on each side of the && must be true for the compound condition to be true. Otherwise the compound condition is false.

|  |  |  |
| --- | --- | --- |
| && operator truth table | | |
| Left-side condition | **Right-side** **condition** | **Result** |
| TRUE | TRUE | TRUE |
| TRUE | FALSE | FALSE |
| FALSE | TRUE | FALSE |
| FALSE | FALSE | FALSE |

● **|| (or) operator:** the conditions on each side of the || must be false for the compound condition to be false. Otherwise the compound condition is true.

|  |  |  |
| --- | --- | --- |
| || operator truth table | | |
| Left-side condition | **Right-side condition** | **Result** |
| TRUE | TRUE | TRUE |
| TRUE | FALSE | TRUE |
| FALSE | TRUE | TRUE |
| FALSE | FALSE | FALSE |

● **! (not) operator:** the value of the condition after the operator is reversed.

|  |  |
| --- | --- |
| NOT operator truth table | |
| Condition | **Result** |
| TRUE | FALSE |
| FALSE | TRUE |

**Short-circuit evaluation**

● Short-circuit evaluation occurs in compound conditions using && (and) or || (or) operators.

● Conditions are evaluated from left to right.

● When conditions are connected with &&, as soon as a condition evaluates to false, the whole condition is false so evaluation stops.

● When conditions are connected with ||, as soon as a condition evaluates to true, the whole condition is true so evaluation stops.

***if* statement**

● The **if statement** is a conditional statement with a condition and one or more blocks of code.

● A **block** is a group of one or more statements that execute sequentially.

|  |  |  |
| --- | --- | --- |
| <block> | is the same as | statement-1;  statement-2;  …  statement-n; |

● Has several layouts including:

***if* layout 1 – test whether or not to execute a block**

if (<condition>)

{

<block>

}

**● *if* layout 1 example:**

if (age >= 18)

{

System.out.println(“Citizen is old enough to vote!”);

}

***if* layout 2 – test whether to execute one block or another**

if (<condition>)

{

<block>

}

else

{

<block>

}

**● *if* layout 2 example:**

if (number >= 0)

{

System.out.println(“Number is nonnegative.”);

}

else

{

System.out.println(“Number is negative.”);

}

***if* layout 3 – test whether to execute one of several blocks**

if (<condition>)

{

<block>

}

*else if (<condition>)*

*{*

*<block>*

*}*

else

{

<block>

}

The italicized part can be repeated as many times as is needed.

**● *if* layout 3 example:**

if (option == 1)

{

System.out.println(“You selected option 1.”);

}

else if (option == 2)

{

System.out.println(“You selected option 2.”);

}

else if (option == 3)

{

System.out.println(“You selected option 3.”);

}

else

{

System.out.println(“You selected an invalid option.”);

}

***if* layout 4 – test whether or not to execute one of several blocks**

if (<condition>)

{

<block>

}

*else if (<condition>)*

*{*

*<block>*

*}*

The italicized part can be repeated as many times as is needed.

**● *if* layout 4 example:**

if (decade == 1960)

{

System.out.println(“You’re in the 1960s!”);

}

else if (decade == 1970)

{

System.out.println(“You’re in the 1970s!”);

}

else if (decade == 1980)

{

System.out.println(“You’re in the 1980s!”);

}

● Indentation is important when coding a conditional statement. It helps the reader identify the blocks within the statement.

● If a block only has **one** statement, the curly braces are **optional**.

***switch* statement**

● The **switch statement** is a limited alternative to the **if statement**.

● A switch statement has an expression that is tested against one or more cases. The data type of the expression must be byte, short, int, char, or String. It may also be an enumerated type.

● A switch statement may be easier to read when there are more than three conditions to test.

● A switch statement may have a **default case** that will run if all other case tests are false.

***switch* layout 1 – test whether to execute one of several blocks**

**(this is logically the same as Layout 3 above)**

switch (<expression>)

{

case <value>:

<block>

break;

*case <value>:*

*<block>*

*break;*

default:

<block>

break;

}

The italicized part can be repeated as many times as is needed.

**● *switch* layout 1 example:**

int option;

…

switch (option)

{

case 1:

System.out.println(“You selected option 1.”);

break;

case 2:

System.out.println(“You selected option 2.”);

break;

case 3:

System.out.println(“You selected option 3.”);

break;

default:

System.out.println(“You selected an invalid option.”);

break;

}

***switch* layout 2 – test whether or not to execute one of several blocks**

**(this is logically the same as Layout 4 above)**

switch (<expression>)

{

case <value>:

<block>

break;

*case <value>:*

*<block>*

*break;*

}

The italicized part can be repeated as many times as is needed.

**● *switch* layout 2 example:**

switch (decade)

{

case 1960:

System.out.println(“You’re in the 1960s!”);

break;

case 1970:

System.out.println(“You’re in the 1970s!”);

break;

case 1980:

System.out.println(“You’re in the 1980s!”);

break;

}

● Curly braces are not required to enclose a case.

● See **Ifs, switches, and ternary operators** sample application on Blackboard.