**Administrative**

**Today’s session**

*for-each* statement

*break* statement

*continue* statement

*exit* statement

java.lang package

DecimalFormat class

Random number generation

Enum type

Homework 2

**Session Topics**

***for-each* statement**

● A **for-each statement** is used to *read* each member of a collection of values.

● Java collections include:

✓ Array

✓ **List** implementing classes including ArrayList, LinkedList, and Stack.

✓ **Queue** implementing classes including DelayQueue and PriorityQueue.

✓ **Set** implementing classes including HashSet and TreeSet.

✓ Other Java collections

● A for statement uses an **index variable** to loop through a collection.

● A for-each statement uses a **collection variable** to loop through a collection.

● The collection variable takes on each value in the collection.

● A for-each statement only moves forward through a collection.

**●** A for-each statement has syntax:

for (<data-type> <collection-member> : <collection>)

{

<block>

}

● See **For loops** sample application on Blackboard.

***break* statement**

● A **break statement** performs an early exit from a switch, while, do-while, or for statement.

● A break statement causes execution to continue **after** the switch, while, do-while, or for statement.

● A break statement is appropriate in a switch statement but rarely appropriate in other statements.

***continue* statement**

● A **continue statement** performs an early exit from a loop within a while, do-while, or for statement.

● In a while or do-while statement, a continue statement causes execution to continue with the condition.

● In a for statement, a continue statement causes execution to continue with the update.

● A continue statement should rarely be used.

● See **For loops** sample application on Blackboard.

***exit statement***

● An **exit statement** is used to end a program.

● An exit statement has syntax:

System.exit(<return-code>);

Where …

**<return-code>** is an integer. 0 generally means the program ended without error, while <>0 means the program ended with error.

● The operating system running the program could then query the return code. In Windows, the code is stored in environment variable %errorlevel%.

● *exit* example:

Scanner keyboard = new Scanner(System.in);

int miles;

System.out.print("Enter the number of miles run: ");

i = keyboard.nextInt();

if (i < 0)

{

System.out.println("Error: you can’t run backwards!");

System.exit(-1);

}

else

{

System.out.println("You must be worn out!");

}

***java.lang* package**

● The **java.lang package** contains some of the most useful Java classes.

● java.lang classes contain code that perform some common tasks – code that we don’t have to write.

● Classes in the java.lang package are the only ones that do not need to be explicitly imported (no **import java.lang.\*;** statement is needed near the top of a program).

● Here are some of the java.lang classes:

| Class | Description |
| --- | --- |
| Wrapper classes | |
| Byte | Wrapper for the *byte* primitive data type. |
| Short | Wrapper for the *short* primitive data type. |
| Integer | Wrapper for the *int* primitive data type. |
| Long | Wrapper for the *long* primitive data type. |
| Float | Wrapper for the *float* primitive data type. |
| Double | Wrapper for the *double* primitive data type. |
| Boolean | Wrapper for the *boolean* primitive data type. |
| Character | Wrapper for the *char* primitive data type. |
| Other classes | |
| Math | Container for methods that perform basic numeric operations including arithmetic, exponential, logarithm, square root, and trigonometric functions. |
| Number | Parent class of classes BigDecimal, BigInteger, Byte, Double, Float, Integer, Long, and Short. |
| Object | Parent class of entire Java class hierarchy. |
| Runtime | Container for methods to enable a program to interface with the environment in which it is running. |
| StrictMath | Same as Math but methods return more precise results. |
| String | Container for methods that manipulate strings and convert values to strings. |
| System | Container for methods to access the standard input (keyboard), standard output (screen), and error output, environment variables. |
| Thread | Container for methods to control execution threads. |

● See **Math fields and methods** sample application on Blackboard.

***DecimalFormat* class**

● The **DecimalFormat class** is an alternative to the number formatting provided by format specifiers used in a printf statement.

● A DecimalFormat object is created with a pattern for formatting numbers.

**●** A DecimalFormat object declaration has syntax:

**DecimalFormat <format-name> = new DecimalFormat("<pattern>");**

Where <pattern> is a set of characters from the following table.

|  |  |
| --- | --- |
| Symbol | Purpose |
| 0 | Show a digit regardless if the number has one at that place. |
| # | Show a digit if the number has one at that place. |
| . | Show the decimal point. |
| , | Separate groups of digits with a comma. |
| E | Separate the mantissa and exponent for exponential formats. |
| ; | Separate multiple formats. |
| % | Multiply the number by 100 and show it as a percentage. |

● The **format method** is used to format a numeric expression as a string.

● The format method has syntax:

**… <format-name>.format(<numeric-expression>) …**

● See **Decimal formatting** sample application on Blackboard.

**Random number generation**

● Random number generation is useful for providing test input to an application.

● There are two ways to generate a sequence of random numbers:

✓ Using method **Math.random()**.

✓ Using class Random and its methods.

● Numbers may be generated in a range.

● Method Math.random() returns a random real number in the range 0 to 1 (but not including 1). This is then scaled to the desired range.

● Random class has several methods including:

✓ **nextDouble()** returns a random real number in the range 0 to but not including 1.

✓ **nextInt()** returns a random integer in the range -2,147,483,648 to 2,147,483,647.

✓ **nextInt(<upper-bound>)** returns a random integer in the range 0 to but not including <upper-bound>.

● See **Random number generation** sample application on Blackboard.

**Enum type**

● An **enumerated type** enables the developer to create their own range of values for a variable.

● An enumerated type may be useful to restrict the range of string values for a variable.

● An enumerated type may only be declared at the same level as a field or method.

● See **Enumerated types** sample application on Blackboard.

**Homework 2**

● Assigned today.

● Available on Blackboard.

● Due in one week.