FIT1051 Notes

2017 GUIDEJava & Programming Essentials

Contents:

BASICS 2

CLASSES 3

STRUCTURE 4

METHODS 5

DRIVER CLASSES 6

PRIMITIVE DATA TYPES 7

VARIABLES AND CONSTANTS 8

OPERATORS AND OPERATIONS 9

STRINGS 11

MEMORY STORAGE 12

ARRAYS 13

SELECTION CONTROL STRUCTURES 14

REPITITION CONTROL STRUCTURES 16

METHODS AND MODULARISATION 18

CLASSES (ADVANCED) 20

ENUMERATORS 22

INHERITENCE 23

POLYMORPHISM 24

INTERFACES 25

Operators and Operations:

- **Operators** are tools to manipulate data, and include the use of mathematics with numbers.
- Expressions evaluate to a single value, which has it's own data type. **Expressions** can perform arithmetic calculations, perform string manipulations, assign values to variables or compare data values.
- When there are more than one operator used, there is an operator order which is similar to BIDMAS in mathematics.
- An assignment statement is just an expression terminated by a semi-colon:

```
myWindow.writeOutLine(testInt = 123);
```

Operation	Operator	Examples	Returned Value
Addition	+	1 + 2	3
Subtraction	-	1 - 2	-1
Multiplication	*	1.5 * 2.0	3.0
Division	/	1.0 / 2.0	0.5
Remainder	%	8 % 3	2
Auto Increment	++	Adds 1 to value	
Auto Decrement	--	Subtracts 1 from value	
Compound Assignment	+=, -=, *=, /=, %=	Adds a value to variable	

- If both operands of a division are integer, then the division will produce an integer number without the fraction. To divide two integers and receive a decimal, .0 must be added at the end of the numbers.

```
myWindow.writeOutLine(1 / 2);      //0
myWindow.writeOutLine(1.0 / 2.0);  //0.5
```

- A **increment** is denoted by ++ and changes the value of an integer by 1. Re.
- A **decrement** is denoted by -- and subtracts one from the integer and has post-decrements and pre-decrements as well.
- In expressions, the difference between post and pre increments/decrements becomes apparent: In the table below, the variable *i1* is set to 1 and the variable *i2* is set to 2.

Expression	Returned Value	i1 after	i2 after	Explanation
i1+ ++i2	4	1	3	i2 is incremented from 2 to 3 immediately, then added to i1
i1+ i2++	3	1	3	i2 is first added to i1, then incremented from 2 to 3
i1 + --i2	2	1	1	i2 is decremented from 2 to 1 immediately, then added to i1
i1 + i2--	3	1	1	i2 is first added to i1, then decremented from 2 to 1

- If possible, avoid using increments and decrements in expressions, and keep them seperate as it can cause confusion.
- **Compound Operations** are an easier way of adding or changing a value of a variable:

```
int i = 10;
```

Arrays:

-
- Each element in the array has an **index** value and array values begin at an index of 0.
- Arrays have their data stored inside the heap.
- Arrays are defined using the [ ] and must be given a starting amount of cells and this is difficult to change later:

```
int [ ] ar = new int[10];
```

- If we created an array with 10 values, the last index available will be 9, as the first value is at index 0. It is easy to assign values to specific indexes in arrays, but very difficult to remove indexes and organise properly:

```
ar[4] = 6;        //Set the 5th element to a value of 6
```

- The array has a property called **length** which is similar to the String.length() method, but in arrays it is just a property. It has an integer data type:

```
int len = ar.length;      //The value will be 10
```

- When creating an array of objects, the actual objects are not initialised until they have their own initialisation:

```
Person [ ] persons = new Person[20];  //Only one object referenced
persons[0] = new Person();    //Now there are two objects referenced
```

- If the initial values are known in advance, then a set of braces can be used to create an identity matrix:

```
int [ ] ar = {2, 5, 8};     //Creates an array with 3 elements
```

- Java's arrays can only do so much and are difficult to manipulate once initialised. Instead, a java library called ArrayList is used. This must be imported by:

```
import java.util.ArrayList;
```

- ArrayLists do not use primitive types but can create an array of any type in other forms (int —> Integer). The Integer class is an advanced version of int and can create an Integer Class.

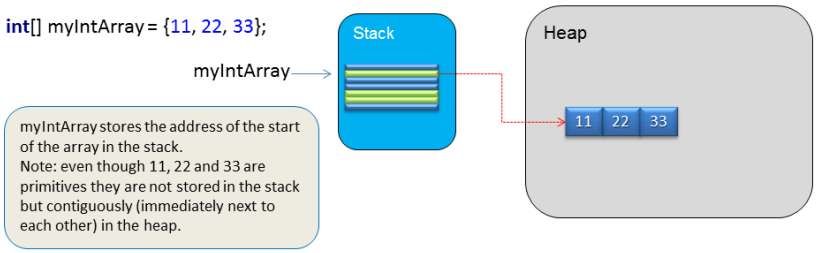
```
ArrayList<Integer> myList = new ArrayList<~>(100);
```

- ArrayLists do not have to have a specified length and can be manipulated at run time:

```
myList.add(3);  //Adds the value of 3 to the next available cell
myList.add(0, 4);                  //Adds the value of 4 to the 0th indexed cell
myList.set(2, 7);          //Changes the value of the 2nd indexed cell to 7
myList.remove(0);        //Removes the cell at the 0th index
```

- Although we set the size to 100 in this example, this is not a fixed value as ArrayLists can be changed and manipulated. It is not required to put any value at all - it helps java for memory storage but does not affect the array's function. To find the size, use the size() method.

```
int i = myList.size();
```



Repetition Control Structures:

- A **repetition control structure** allows a block of code to be repeated a given amount of times, or until a condition is met.
- If the number of repetitions is known, a counter control loop will be used. If the number of repetitions is not known, a sentinel or a value controlled loop is used.
- A **for** loop is an example of a **counter controlled loop** and repeats a code block a specific number of times given in it's parenthesis.
- A for loop has 3 statements within the parenthesis, one which initialises the variable of which is repeating, one which sets the condition to which it repeats by, and one which increments it:

```
for (int i = 0; i < 6; i++) {
        //Run repeated statements
}
```

- In this case, the loop will loop through the values of 0, 1, 2, 3, 4 and 5 in the i variables. It will not run the loop when the value of i is equal to 6.
- An **while loop** repeats a block of code indefinitely until a condition is met. It can do the same function as a for loop as well:

```
int i = 0;
while (i < 6) {
        //Run repeated statements
        i ++;
}
```

- In this case, the condition is checking if the value of i is less than 6 and will run the loop 6 times. However, the program will crash due to an infinite loop if the value of i is not incremented within the loop.
- A **sentinel control loop** has a condition that is compared to a specific value and the loop is run when the condition is false.

```
while (value != SENTINEL)
```

- A **value control loop** has a calculated value in it's condition and the loop is run until a specific value is reached.

```
while (money > 56)
```

- Loops can also be nested within each other and the loop condition values and incremental values can be used within the coding block.

```
for (int i = 0; i < 10; i++) {
        System.out.print("Outer loop value of" + i);
        for (int j = 0; j < 5; j++)
                System.out.print("Inner loop value of" + j);
}
```

- In this loop case, the outer loop will run 10 times and the inner loop will run 5 times per outer loop running. This means that the inner loop will run a total of 50 times.
- An **enhanced for loop** can be used to repeat through all the elements of an array:

```
for (type elem : array)
```

- There are two ways of looping through an array elements. The first uses a normal for loop with a condition:

```
String [ ] myArray = {"One","Two","Three","Four","Five"};
for (int i = 0; i < myArray.length; i++)
        System.out.println(myArray[i]);
```

- The second will use an enhanced for loop:

```
String [ ] myArray = {"One","Two","Three","Four","Five"};
for (String elem : myArray)
```

Classes (Advanced):

- Any **concept** class created is useless without a **driver** class which contains to main method that runs on program execution.
- The **toString()** method is used to report the state of an object. It is overridable and is available to all classes by default. It takes in no parameters and will return a String. The toString method is part of a Java class called “Object” and thus is connected to all objects, but for greater accuracy, the toString method can be coded in each specific class to allow the state dump to show the correct amount of values.
- To code the toString method, we much **override** it. Overriding is when we take precedence over an inherited version and occurs when both methods have an **identical signature** and the same return type as the inherited method.
- A **method signature** is it’s name and the ordered list of the data types of it’s parameters. We can override the toString method if the two methods have the same signature:

```
int height, width;
public  String toString() {
        String retVal = "Height:" + height + ",Width:" + width;
        return retVal;
}
```

- Now when the toString method is called on this specific class, the values of it’s instance variables height and width are returned.
- Method **overloading** can occur when two methods in the same class have the same name but a different ordered list of data types of it’s parameters. (same name but different signatures). For example, we can create two methods that do the same function, but with seperate inputs:

```
public int AddNumbers (int i, int j) {
        return i + j;
}
public int AddNumbers (int i, int j, int k) {
        return i + j + k;
}
```

- In the two methods above, the ordered list of the parameters are different which means the two methods are overloads of each other. When calling the specific function, the one that matches the exact parameter list will be run:

```
int sum = AddNumbers (2, 3, 5);          //The first method will be called
```

	Methods Involved	Method Name	Ordered List of Data Types of Parameters	Result
Overloading	Methods in same class	Same	Must be different	More than one method with the same name in a class is possible
Overriding	Methods in a class and in another class that inherits these methods	Same	Must be the same	The inherited method is blocked by the overriding method

- An example of overloading is the *Color* class which can accept three or four parameters. If three double parameters are inputted, the values of r, g and b are given. The fourth parameter can be used to specify the alpha value, but if left out it is assumed to have a value of 1.