CSC 20 – Program Concepts And Methodology II Midterm Exam Study Guide

1. Know The Java Fundamentals For Programming: The Parts Of A Java Program, The Scanner Class And System.Out.Println Method, Variables And Literals, Primitive Data Types (Byte, Short, Int, Double, Etc) And Reference Types (Array, Strings), And Operators.

Know the Java Fundamentals for Programming:

- **The Parts of a Java Program**
  - main() method
    - java interpreter starts by calling the public class's main() method.
    - main() method takes an array of String as an argument
    - main() method must be declared public, static and not return a value (void).
    - Signature of the main() method can be any of these:
      - public static void main(String args[])
      - public static void main (String [] args)
      - static public void main (String [] args)
        - note: args can instead be any valid identifier like "anything"
- **The Scanner Class and System.Out.Println Method**
  - Scanner class provides methods for reading byte, short, int, long, float, double, and String data types from the Java console or text files.
  - Scanner is in the java.util package.
  - Scanner parses(separates) input into sequences of characters called tokens.
  - By default, tokens are separated by standard white space characters (tab, space, newline, etc).
- **Variables and Literals**
- **Primitive Data Types (Byte, Short, Int, Double, Etc.)**
  - byte (8 bits), short (16 bits), int(32 bits), long (64 bits)
  - float (32 bits), double (64 bits)
  - char -Unicode! e.g., '\u12ab' (16 bits)
  - Boolean(16 bits, true/false)
- **Reference Types (Array, Strings) AKA Subtypes of Objects**
  - Classes: String etc.
  - Arrays
- **Operators**

Operator	Description	Example
+ (Addition)	Adds values on either side of the operator.	A + B will give 30
- (Subtraction)	Subtracts right-hand operand from left-hand operand.	A - B will give -10
*** (Multiplication)**	Multiplies values on either side of the operator.	A * B will give 200

/ (Division)	Divides left-hand operand by right-hand operand.	B / A will give 2
% (Modulus)	Divides left-hand operand by right-hand operand and returns remainder.	B % A will give 0
++ (Increment)	Increases the value of operand by 1.	B++ gives 21
-- (Decrement)	Decreases the value of operand by 1.	B-- gives 19
== (equal to)	Checks if the values of two operands are equal or not, if yes then condition becomes true.	(A == B) is not true.
!= (not equal to)	Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.	(A != B) is true.
> (greater than)	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.	(A > B) is not true.
< (less than)	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.	(A < B) is true.
>= (greater than or equal to)	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.	(A >= B) is not true.
<= (less than or equal to)	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.	(A <= B) is true.
&& (logical and)	Called Logical AND operator. If both the operands are non-zero, then the condition becomes true.	(A && B) is false
|| (logical or)	Called Logical OR Operator. If any of the two operands are non-zero, then the condition becomes true.	(A || B) is true
! (logical not)	Called Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true, then Logical NOT operator will make false.	!(A && B) is true
=	Simple assignment operator. Assigns values from right side operands to left side operand.	C = A + B will assign value of A + B into C
+=	Add AND assignment operator. It adds right operand to the left operand and assign the result to left operand.	C += A is equivalent to C = C + A
-=	Subtract AND assignment operator. It subtracts right operand from the left operand and assign the result to left operand.	C -= A is equivalent to C = C – A
*=	Multiply AND assignment operator. It multiplies right operand with the left operand and assign the result to left operand.	C *= A is equivalent to C = C * A
/=	Divide AND assignment operator. It divides left operand with the right operand and assign the result to left operand.	C /= A is equivalent to C = C / A
%=	Modulus AND assignment operator. It takes modulus using two operands and assign the result to left operand.	C %= A is equivalent to C = C % A

2. Java Statements: Simple Statements, Compound Statements, Alternative Statements: If, Switch, Repetitive Statements: For, While, Do/Do-While. Know The Ordering Of Operator Precedence &

Associativity. Given An Expression, Evaluate Its Outcome’s Value. Know Data Conversion Rules (Widening And Narrowing).

WEEK 2

Java Statements

Simple Statements	

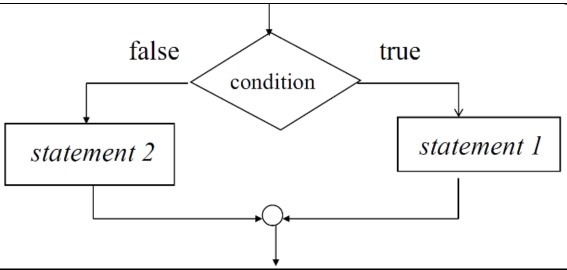
`expression;`
`// A simple statement requires a`
`//statement terminator AKA the`
`//semicolon.`
	What is an expression?
	“Any combination of operands and operators, which, when evaluated, yields a value”

Compound Statements	

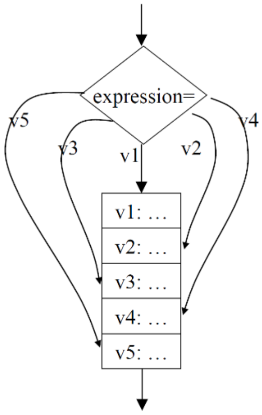
FORMAT
`{`
`    s1;`
`    s2;…;`
`}`
	• A sequence of zero or more statements contained between “ **{** ” and “ **}** ”
	• A compound statement is also called a block.
	• A compound statement is considered as a single statement.

- **Alternative Statements: If, Switch**

If Statements (Two forms)	

ONE
`if  (condition)  statement`
TWO
`if  (condition)  statement1`
`else  statement2`
	

Switch Statement	

FORMAT
`switch (score / 10) {`
`    case 10:`
`    case  9: grade = ‘A’; break;`
`    case  8: grade = ‘B’; break;`
`    case  7: grade = ‘C’; break;`
`    case  6: grade = ‘D’; break;`
`    default: grade = ‘F’;`
`}`
	 expression=
	v5
	v3
	v1
	v2
	v4
	v1: …
	v2: …
	v3: …
	v4: …
	v5: …

- **Repetitive Statements: For, While, Do/Do-While**

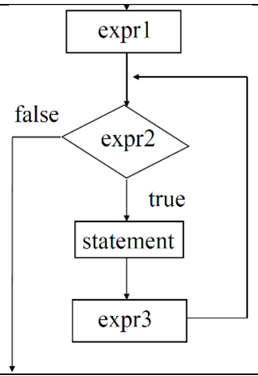
The For Statement

FORMAT

```
for (expr1; expr2; expr3) statement
```

EXAMPLE

```
for(count = 1; count <= limit; count++){
     inti  = stdin.nextInt();
     sum += i;
}
```



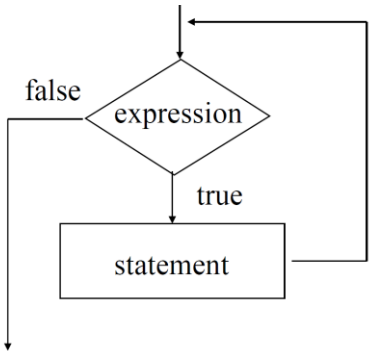
The While Statement

FORMAT

```
while (expression) statement
```

EXAMPLE

```
count = 1;
limit = 10;
sum = 0;
while(count <= limit){
     inti = stdin.nextInt();
     sum += i;
     count++;
}
```



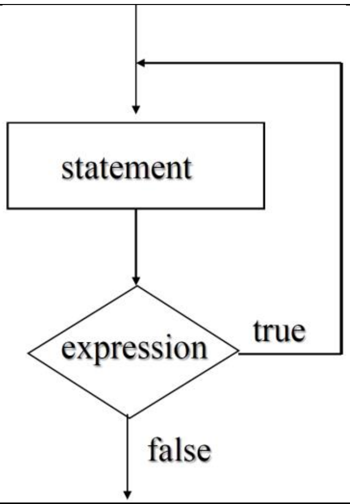
The Do-While Statement

FORMAT

```
do {
     statements
} while (expression);
```

EXAMPLE

```
count = 0;
do {
     inti = stdin.nextInt();
     sum += i;
     count++;
} while (count <= 10);
```



- **Know the Ordering of Operator Precedence & Associativity**
    - Just know all are **LEFT TO RIGHT**, but Unary, Conditionals, and assignment operators

Category	*Operator*	*Associativity*
Postfix	>() [] . (dot operator)	Left to right
Unary	>++ - - ! ~	Right to left
Multiplicative	>* /	Left to right
Additive	>+ -	Left to right
Shift	>>> >>> <<	Left to right
Relational	>> >= < <=	Left to right
Equality	>== !=	Left to right
Bitwise AND	>&	Left to right
Bitwise XOR	>^	Left to right
Bitwise OR	>|	Left to right
Logical AND	>&&	Left to right
Logical OR	>||	Left to right
Conditional	?:	Right to left
Assignment	>= ,+=, -=, *=, /=, %=, >>=, <<=, &=, ^=, |=	Right to left

- **Given an Expression, Evaluate Its Outcome's Value**
- **Know Data Conversion Rules (Widening and Narrowing)**

Code Design and Development

- **Be Able to Explain and Apply the Stepwise Refinement Process (AKA Top-Down Design)**

The basic idea is to repeatedly decompose pseudocode statements until each pseudocode statement can be coded in a couple of programming language statements.

THE ADVANTAGE

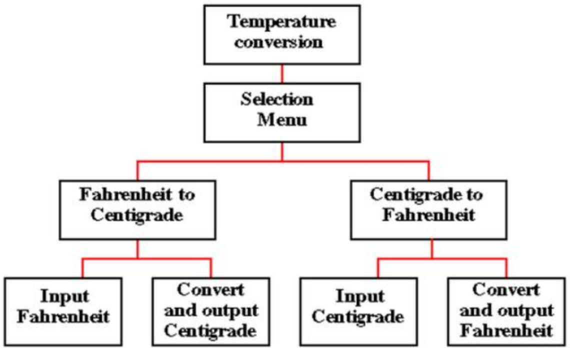
- Breaking the problem into parts allows more than one person to work on the solution.
- Breaking the problem into parts helps us to clarify what needs to be done.
- Parts of the solution may turn out to be reusable.
- At each step of refinement, the new parts become less complicated and, therefore, easier to figure out.

In short

- We break the problem into parts
- Then break the parts into parts
- Soon, each of the parts will be easy to do

A graphical example

Design and create an Java program that, given a temperature in Centigrade converts it to Fahrenheit and vice-versa.



- **Know How to Work with Coding Optimization Scenarios**

<table>
<tr><th colspan="2">Coding Optimization</th></tr>
<tr><td>• Code optimization is any method of code modification to improve code quality and efficiency.
• A program may be optimized so that it becomes a smaller size, consumes less memory, executes more rapidly, or performs fewer input/output operations.
• Sometimes, these are tradeoffs (i.e. performance vs. readability)</td><td><b><u>TYPES</u></b>
<b>Intermediate code level</b>
• We are looking at this part now
<b>Machine code level</b>
• Instruction selection, register allocation, etc.</td></tr>
</table>

4. Array and String: Declaration and Initialization. Use Assignment Operator with Array. Array Cloning, And Equality. String Concepts, Declarations, And Operators.

Array and String

- **Array and String:**

<table>
<tr><th colspan="2">Array</th></tr>
<tr><th colspan="2">Declaring an array does not create it! No memory is allocated for individual array elements. This requires a separate creation step.</th></tr>
<tr><th>Declaration (2 ways)</th><th>Initialization</th></tr>
<tr><td rowspan="2"><b><u>ONE</u></b>
datatype[] arrayname1, arrayname2;
<b>Example:</b> int[] myArray1, myArray2;
<b><u>TWO</u></b>
Datatype arrayname[];
<b>Example:</b> int myArray1[], x, myArray2[];</td><td>arrayName = newdatatype[arraySize];
<b>Example:</b> myList = new double[8];</td></tr>
<tr><td><b>NOTE:</b> <i>The new keyword creates an object or array. The object or array is created in a location of memory called the heap. A reference (pointer) to the array is assigned to the variable.</i></td></tr>
</table>

- **Use Assignment Operator with Array**
- **Array Cloning, And Array Equality**
- **String Concepts, Declarations, And Operators**

5. Java Classes: Classes and Object, Instance Fields And Methods, Constructors, Overloading Methods And Constructors. Package and Import Statements. Passing Objects As Arguments To Methods.

Java Classes

- **Classes and Object**
- **Instance Fields and Methods**
- **Constructors**
- **Overloading Methods and Constructors**

- **Package and Import Statements**
- **Passing Objects as Arguments to Methods**

6. Inheritance: Define Inheritance, Calling The Superclass Constructor, Overriding Superclass Methods. Know Two Access Specifications Within A Class: Private And Public. Know How To Write A Setters/Getters Methods.

Inheritance

- **Define Inheritance**
- **Calling the Superclass Constructor**
- **Overriding Superclass Methods**
- **Know Two Access Specifications Within A Class:**
- **Private and Public**
- **Know How to Write A Setters/Getters Methods.**

7. Linked Lists: Understand Concepts. Familiar With Operations On Linked List (Traversing, Addfirst, Addlast, Insertbefore, Insertafter, Backward, Forward, Etc). Concepts Of A Node Stored As An Object (I.E Csusstudent) With Pointer.

Linked Lists: Understand Concepts.

- **Familiar with Operations On Linked List**
  - Traversing
  - Addfirst
  - Addlast
  - Insertbefore
  - Insertafter
  - Backward
  - Forward
- **Concepts of A Node Stored As An Object (I.E Csusstudent) With Pointer.**