CS 161 Final Review Material

- **Arrays**

  - **Declaration:**

An array can be declared by creating a statement beginning with the data type followed by square brackets '[]' and the desired array name.

```
dataType[] arrayName;
```

For a multi-dimensional array, the data type should be followed by two pairs of square brackets '[][]'.

```
dataType[][] arrayName;
```

  - **Initialization:**

Arrays can be initialized in multiple ways.
*Note: Array element positions range from 0 to one less than the array size, (0, size-1).

Simultaneous memory allocation and initialization:

Memory allocation and initialization can be done simultaneously by assigning a set (denoted by curly brackets '{}' where each element of the set contained within are separated by commas.) to the desired array during declaration.

```
dataType[] arrayName = { 0, 1, 2, 3, 4 };
```

For a multi-dimensional array, assign multiple sets, separated by commas, all within an additional pair of curly braces '{}', to the array.

```
dataType[][] arrayName = { { 0, 1, 2, 3, 4 },
                           { 5, 6, 7, 8, 9 } };
```

Memory allocation:

To allocate memory to an array, a size must be specified. This is done by assigning a new data type object, followed by square brackets containing the desired array size '[size]', to the array.

```
dataType[] arrayName = new dataType[size];

dataType[] arrayName;
arrayName = new dataType[size];
```

For multi-dimensional arrays, assign a new data type object, followed by two pairs of square brackets containing the desired array sizes '[column size][row size]', to the array.

```
dataType[][] arrayName = new dataType[column size][row size];

dataType[][] arrayName;
arrayName = new dataType[column size][row size];
```

Initialization:

To initialize an array (without using the Arrays class), each element must be initialized separately. To do this, assign the desired data value to a specific array element. To reference a specific array element, use the array name, followed by square brackets containing the position of the element within the array '[position]';

```
arrayName[position] = dataValue;
```

For multi-dimensional arrays, we must specify both the row position and column position.

```
arrayName[column position][row position] = dataValue;
```

- **Passing arrays to methods:**

When passing arrays as arguments to methods, only the address of the array is passed. To pass the address of the array, the array name is the only argument used, and the full data type and a new name should be accepted as parameters.

```
public static void main( String args[] ) {
        dataType[] arrayName = new dataType[size];
        methodName( arrayName );
}
public void methodName( dataType[] parameterName ) {
        // method content
}
```

Passing an array in a return statement is handled similarly.

```
public void methodName() {
        dataType[] arrayName = new dataType[size];
        return arrayName;
}
public static void main( String args[] ) {
        dataType[] returnName = methodName();
}
```

When passing a single element of an array, append the array name with square brackets containing the position of the element desired to be passed as an argument '[position]'. To receive a specific element as a parameter to a method, specificy the data type of the element followed by a parameter name.

```
public static void main( String args[] ) {
        dataType[] arrayName = new dataType[size];
        methodName( arrayName[position] );
}
public void methodName( dataType parameterName ) {
        // method content
}
```

- **Behavior of good programs (Efficient, Effective, Robust)**
  - **Efficiency:**

Efficiency describes how well your program performs its tasks. For example, how fast does your program perform its tasks? How much memory is used? A fully efficient program will take up minimal space in memory when executed, execute at optimal speeds, and be as minimally demanding on the processor of the computer running the program.

  - **Effectiveness:**

Effectiveness describes the ability of your program to perform its tasks. For example, does your program do (in the desired manner) what you intended it to? A fully effective program will be able to perform all required tasks exactly as described in the task description.

  - **Robustness:**

Robustness describes how reliable your program is. For example, how does it handle extreme workloads, bad or unpredictable user inputs? A fully robust program should never crash and should always act as intended, under any conditions.

- **Error handling techniques**

Users providing unexpected input can often cause programs to run in unintended manners. To accommodate for this, it is necessary to implement error handling within a program. Often errors are found where: objects and variables interact with each other, the outside world is accessed, or there is user input. While loops with an if statement contained within are often used as the most efficient means of error handling simple user input.

The while loop which will contain user input prompts is used to ensure input read from the Scanner object is within specified parameters. To ensure we trigger the while loop, we must initialize the variable, which will contain the user input, to an invalid value.

The if statement within the while loop is used to ensure the Scanner is provided with the correct data type, before assigning the input to an actual variable of that data type. The if statement will prevent exceptions stemming from trying to assign a String to a double, or similar complications.

These complications are essential to consider while designing a program, as well as when writing the program. As beginning to accommodate for errors, after implementing a design, may require starting over with new code.

We must also initialize the scanner object every repetition of the loop to ensure fresh input is provided. Another if statement can be provided within the while loop, with the same control arguments as the while loop, to print erroneous input messages to the user.

Some examples are demonstrated below:

__Example - Reading a double between 0 and 100:__

```
Scanner input;
double value = -1;
while( value < 0 || value > 100 ) {
        System.out.println( “User, please input a double” );
        input = new Scanner( System.in );
        if( input.hasNextDouble() )
                value = input.nextDouble();
        if( value < 0 || value > 100 )
                System.out.println( “Invalid input” );
}
```

__Example - Reading a character selection from a list containing the set { u, d, l, r }:__

```
Scanner input;
String list = “udlr”;
String value = “”;
while( !( list.contains(value) && list.length() == 1 ) ) {
        System.out.println( “User, please select a direction [ u, d, l, r ]:” );
        input = new Scanner( System.in );
        if( input.hasNextLine() )
                value = input.nextLine();
        if( !( list.contains(value) && list.length() == 1 ) )
                System.out.println( “Invalid input” );
}
```

- **Methods and classes**
    - **Classes:**

Classes are one of the most basic building blocks of Java programs. Each class represents a data structure with attributes and behaviors.

Declaration:

To make a class declaration, a visibility modifier must be followed by the keyword ‘class’ and subsequently followed by the class name and curly brackets ‘{}’. The contents of the class (attributes and behaviors) are contained within the curly brackets. By convention, class names begin with a capital letter, and every separate word in the class name will also be capitalized.

```
visibilityModifier class ClassName {
        // class content
}
```

Instantiations of a class:

Java is an object-oriented programming language. Each class in Java can be instantiated as an object. An instantiation of a class is called an object. Each instantiation of a class is a unique object, containing unique attributes. When attributes of one instantiation are modified, other instantiations are not affected.

A class can be instantiated by creating a statement beginning with the class name, followed by the object (or instance) name, and assigning it (using the keyword ‘new’) the class name followed by parentheses ‘()’.

```
ClassName objectName = new ClassName();
```

Once a class has been instantiated as an object, the object’s attributes and methods can be accessed non-statically by following the objectName with a period ‘.’ and the specific attribute or method desired.

```
System.out.print( objectName.value );
objectName.methodName();
```

To make reference to static attributes or methods of a class (without using an instantiation of a class), use the class name followed by a period ‘.’ and the specific attribute or method desired. Only static attributes and methods can be referenced in this way. No modification can be made to the content of a class statically.

```
System.out.print( ClassName.value );
ClassName.methodName();
```

- **Methods:**

Methods describe behaviors of a class, or actions which can be performed. Methods contain statements which dictate the actions performed during execution. Methods can accept arguments as parameters, manipulate this data, and return the altered data. The driving class will contain a main method that is automatically called.

Declaration:

To make a method declaration, a visibility modifier must be followed by the keyword ‘static’ (if universality is desired), then a return data type (void is used for no return type), the method name, parentheses containing parameter types and names (left empty if no parameters are used), and finally curly braces ‘{}’. The content of the method is contained within the curly braces. By convention, method names begin uncapitalized, with every other word following in the name capitalized.

```
public static dataType methodName() {
        // method content
}
private void methodName( dataType parameterName ) {
        // method content
}
```

- **Visibility modifiers**

Visibility modifiers (Access level modifiers) determine whether other classes can use a particular field or invoke a particular method.

Modifier	Class	Package	Subclass	World	Java Example
public	X	X	X	X	public void methodName()
protected	X	X	X		protected void methodName()
(no modifier) / ‘Phriendly’	X	X			void methodName()
private	X				private void methodName()