Review 1 (Modules 1 - 5)

Variables

Variables hold data. They can store data that we write as part of our code, data that the user types into your program, data that comes from a database in another computer, etc.

Declaring a variable is simple. All you need to know is the type, what you want to name it, and the value it should hold:

```
String myName = "Marcos Minond";
boolean isAlive = true;
int age = 92;
```

Remember the parts of a variable: "type", "name", "=", "value".

Arrays

Arrays are lists of values. They group together values/data that should be grouped. They are simply lists:

```
String[] friends = {
  "Marcos Minond",
  "Mohamed Alsoudani",
  "Ryan Moore"
};
```

Above is a sample array of `String` values. To declare an array in Java, I take any type (`int` , `short` , `String` , `boolean` , etc.) and add a `[]` after the type, I give it a name just like I would with any other type, then put the list of comma separated values inside the `{}` brackets.

Accessing a specific entry in the array is done by placing `[N]` after the variable name, where `N` is the index of the value:

```
System.out.println(friends[1]);
```

This will print out the second item (because arrays start at 0, so 1 is the second item) in my `friends` array, which if you look at the list, you'll see it includes Mohamed's name.

To get the "length" of an array, which in the case of arrays it is the number of items it holds, check the `.length` property: `friends.length;`

Control structures - `if` statements

There are two types of control structures in Java. The first kind allows you to run parts of your code when a certain condition is met, like when a users guesses the right number, or when they type in the correct password, etc. There are also control structures that allow your code to run multiple times in a row. This is usually done when you need to take the same action multiple times. Arguably, the two most important control structures in Java are the `if` and `for` statements.

`if` statements say, if something is true, then do this, otherwise check this other thing and do that instead. `if (conditionIsTrue) doThis(); else doAnotherThing();` :

```
int myNumber = 42;

if (myNumber == 0) {
  // code
}

if (myNumber == 0) {
  // code
} else {
  // code
}

if (myNumber == 0) {
  // code
} else if (myNumber == 100) {
  // code
} else {
  // code
}

if (myNumber == 0) {
  // code
} else if (myNumber == 50) {
  // code
} else if (myNumber == 100) {
  // code
} else {
  // code
}
```

You can have a single `if` statement or a group. A group must always start with `if` statement, it can have as many `else if` statements, and only one `else` statement at the very end of the group. The `else if` and `else` statements are optional.

Below is an example of reading input from a user and comparing it to a secret code that is stored in our code:

```
import java.util.Scanner;

public class GuessTheSecretCode {
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    String secret = "hello";

    System.out.printf("What is the secret? Type your secret and press enter: ")
    String userInput = sc.next();

    if (userInput.equals(secret)) {
      System.out.println("You guessed the correct secret!");
    } else {
      System.out.println("Incorrect guess.");
    }
  }
}
```

The code above reads asks the user to input a secret which is then read with `sc.next()`. This function is a method of the `Scanner` class that we can use because we have `import java.util.Scanner;` above our class declaration. The user's input is then compared to the known secret code using `.equals`, which is how we compare strings. If the two strings are equal, we then tell the user that they guessed the correct secret code. And if they are not, we tell them that they guessed incorrectly.

Below is a code example of how you would do something similar, but with numbers instead:

```
System.out.printf("Type a number and press enter: ");
Scanner sc = new Scanner(System.in);

int numberUserIsSupposedToGuess = 42;
int numberUserGuessed = sc.nextInt();

if (numberUserGuessed == numberUserIsSupposedToGuess) {
  System.out.println("The two numbers are the same.");
} else if (numberUserGuessed > numberUserIsSupposedToGuess) {
  System.out.println("The number you guessed is more than the number you are su
} else {
  System.out.println("The number you guessed is less than the number you are su
}
```

In the code above we are first checking if the two numbers are the same. Since we are working with numbers we can do this with the `==` operator. Then we check if the number guessed is larger than the number the user is supposed to guess using the `>` operator. Once we know a number is *not* greater than (`>`) and *not* equal to (`==`) another number, the only option we are left with is that it is less than (`<`) so we do not need a comparison here.

Control structures - `for` loop statements

Another control structure, `for` loops have two forms, but I'll only mention one in this handout. Here is an example:

```
String[] friends = {
  "Marcos Minond",
  "Mohamed Alsoudani",
  "Ryan Moore"
};

for (int i = 0; i < friends.length; i++) {
  // loop body
  System.out.println(friends[i] + " is my friend.");
}
```

This code creates an array of `String` values then loops over them and prints out a message. The different parts of the for loop are the following:

-. Setup: `int i = 0` -. Check: `i < friends.length` -. Increment: `i++`

Read for loops like this:

1. Declare `int i` and set its value to `0` .
2. Check if `i` is less than then number of items in the `friends` array (using `friends.length`), if it is, go to step #3. If it is not, go to step #5.
3. Run the code that is in the body of the loop.
4. Increment `i` by one and go back to step #2.
5. Execute the code that is after/below the `for` loop.

Class properties

Think of classes as a way to group different parts of a larger object into a single object so that we can treat it as a single unit. One of the features classes have is the ability to have properties. Properties are very similar to variables, expect that they are tied to the class:

```
class Movie {
  String title;
  int releaseYear;
}
```

I can use this movie class by declaring a variable and giving it a type of `Movie` since classes that you create are type in Java:

```
Movie starWars = new Movie();
starWars.title = "Star Wars";
starWars.releaseYear = 1977;
```

`starWars` is an instance of the `Movie` class. Since the `Movie` class has a `title` and a `releaseYear` property that are both public, I can set them by accessing the property using dot notation: `variableName.propertyName` .

Class methods

Along with properties, classes can have methods. Every class you have created so far as a `state void main(String[] args)` method. Methods are one of the building blocks in Java. You own classes can have methods:

```
class Movie {
  String title;
  int releaseYear;

  String getMovieInformation(String greeting) {
    return greeting + ", " + title + " was released in " + releaseYear;
  }
}
```

By looking at the method declaration `String getMovieInformation(String greeting` I can see the different parts. First, it has a return type of `String`, a name which is `getMovieInformation`, and it take one argument, `String greeting`, that I can use inside of the method's body.

I can use the `getMovieInformation` method very much like I use properties, except that since it is a method call, I can pass arguments to it within `()` parentheses.

```
public class MovieRunner {
  public static void main(String[] args) {
    Movie starWars = new Movie();
    starWars.title = "Star Wars";
    starWars.releaseYear = 1977;

    String info = starWars.getMovieInformation();

    System.out.println(info);
  }
}
```

Class constructors

A constructor is a special method in Java. To create one, all I have to do is declare a method in my class that does not have a return value and has the same name as the class itself:

```
class Movie {
  String title;
  int releaseYear;

  Movie(String t, int r) {
    title = t;
    releaseYear = r;
  }

  String getMovieInformation(String greeting) {
    return greeting + ", " + title + " was released in " + releaseYear;
  }
}
```

This constructor can take any parameters that I can use in whatever ways I want to. One of the typical ways of using constructors is by creating one that take all of the values you need to store in the class. Remember how our `Movie` class has a `title` and a `releaseYear` year? Well instead of setting those two properties separately (see the "Classes properties" section), I can create a constructor that takes both values and stores them in the properties that I need them in:

```
public class MovieRunner {
  public static void main(String[] args) {
    Movie starWars = new Movie("Star Wars", 1977);

    String info = starWars.getMovieInformation("Hello Marcos");

    System.out.println(info);
  }
}
```