Overview

- Review if/else, while
- Conditionals
- For loops
- Arrays
- Combining for loops and arrays
- Methods
- Introduction to Object Oriented Programming
- Common Problems, Style points, & Soln to Assignment 1
if statement

if (condition) {
    statements
}

if statement

if (x = 8) {
    System.out.println("Correct! The number is 8.");
}
if statement

```java
public static void main(String[] arguments) {
    int x = 8;

    if (x == 8) {
        System.out.println("Correct! The number is 8.");
    }
}

// What is the output?
if statement

```java
public static void main(String[] arguments) {
    int x = 8;

    if (x == 8) {
        System.out.println("Correct! The number is 8.");
    }

    x = 9; // x reassigned

    if (x == 8) {
        System.out.println("Correct! The number is 8.");
    }
}
// Output?
```
public static void main(String[] arguments) {
    int x = 8;
    if (x == 8) {
        System.out.println("Correct! The number is 8.");
    } else System.out.println("No, not " + x);

    x = 9;
    if (x == 8) {
        System.out.println("Correct! The number is 8.");
    } else System.out.println("No, not " + x);
}

// Output?
if/else statement

```java
public static void main(String[] arguments) {
    int x = 8;
    if (x == 8) {
        System.out.println("Correct! The number is 8."); // printed
    }
    else System.out.println("No, not " + x);

    x = 9;
    if (x == 8) {
        System.out.println("Correct! The number is 8.");
    }
    else System.out.println("No, not " + x); // printed
}
```
if/else statement

```java
    else System.out.println("No, not " + x);
```

VERSUS

```java
    else {
        System.out.println("No, not " + x);
    }
```
if/else statement

```java
else System.out.println("No, not " + x);
```

VERSUS

```java
else {
    System.out.println("No, not " + x);
    System.out.println("Guess again!");
}
```
class factorial {
    public static void main (String[] arguments) {
        int factorial = 1;
        int i = 1;

        while (i <= 10) {
            factorial = factorial * i;
            System.out.println(i + "! = " + factorial);
            i = i + 1;
        }
    }
}

// Output?
while loop

class factorial {
    public static void main (String[] arguments) {
        int factorial = 1;
        int i = 1;

        while (i <= 10) { // first iteration, factorial = 1, i = 1
            factorial = factorial * i;
            System.out.println(i + "! = " + factorial);
            i = i + 1;
        }
    }
}

// Output?
while loop

```java
class factorial {
    public static void main (String[] arguments) {
        int factorial = 1;
        int i = 1;

        while (i <= 10) { // first iteration, factorial = 1, i = 1
            factorial = factorial * i; // 1 * 1
            System.out.println(i + "! = " + factorial);
            i = i + 1;
        }
    }
}

// Output?
```
```java
class factorial {
    public static void main (String[] arguments) {
        int factorial = 1;
        int i = 1;

        while (i <= 10) { // first iteration, factorial = 1, i = 1
            factorial = factorial * i; // 1 * 1
            System.out.println(i + "! = " + factorial); // 1! = 1
            i = i + 1;
        }
    }
}

// Output?
```
class factorial {
    public static void main (String[] arguments) {
        int factorial = 1;
        int i = 1;

        while (i <= 10) {// first iteration, factorial = 1, i = 1
            factorial = factorial * i; // 1 * 1
            System.out.println(i + "! = " + factorial); // 1! = 1
            i = i + 1; // 1 + 1
        }
    }
}

// Output?
while loop

class factorial {
    public static void main (String[] arguments) {
        int factorial = 1;
        int i = 1;

        while (i <= 10) {
            factorial = factorial * i; // 1 * 2
            System.out.println(i + "! = " + factorial); // 2! = 2
            i = i + 1; // 2 + 1
        }
    }
}

// Output?
while loop

class factorial {
    public static void main (String[] arguments) {
        int factorial = 1;
        int i = 1;

        while (i <= 10) {
            // third iteration, factorial = 2, i = 3
            factorial = factorial * i;  // 2 * 3
            System.out.println(i + "! = " + factorial);  // 3! = 6
            i = i + 1;  // 3 + 1
        }
    }
}

// Output?
factorial class output

1! = 1
2! = 2
3! = 6
4! = 24
5! = 120
6! = 720
7! = 5040
8! = 40320
9! = 362880
10! = 3628800
Quiz Time!

Identify which of the following while loops will:

- Execute indefinitely
- Execute a finite amount of times
- Never execute
boolean jamieFoxxIsSinging = true;
while (jamieFoxxIsSinging) { // defaults to == true
    System.out.println("She give me money / When I'm in need"); }

boolean laRouxIsBulletproof = true;
while (laRouxIsBulletproof == false) {
    System.out.println("I'll let you sweep me off my feet"); }

boolean jamieFoxxIsSinging = true;
while (jamieFoxxIsSinging) { // defaults to == true
    System.out.println("She give me money / When I'm in need"); } // loop never finishes!

boolean laRouxlsBulletproof = true;
while (laRouxlsBulletproof == false) {
    System.out.println("I'll let you sweep me off my feet");
} // loop never executes
Quiz Time!

```java
int countdown = 10;
while (countdown > 0) {
    System.out.println(countdown + "...");
    countdown = countdown + 1;
}
System.out.println("Blast off!");

int knockTimes = 3;
while (knockTimes > 0) {
    System.out.println("Knock!");
    knockTimes = knockTimes - 1;
}
```
Quiz Time!

```java
int countdown = 10;
while (countdown > 0) {
    System.out.println(countdown + "...");
    countdown = countdown + 1; // should be - not +
} // loop never finishes
System.out.println("Blast off!");

int knockTimes = 3;
while (knockTimes > 0) {
    System.out.println("Knock!");
    knockTimes = knockTimes - 1;
} // loop finishes after 3 knocks
```
Comparison operators

\(x > y\): \(x\) is greater than \(y\)

\(x < y\): \(x\) is less than \(y\)

\(x \geq y\): \(x\) is greater than or equal to \(y\)

\(x \leq y\): \(x\) is less than or equal to \(y\)

\(x == y\): \(x\) equals \(y\)

Q: What is the difference between \(==\) and \(=\)?
Comparison operators

\( x > y \): \( x \) is greater than \( y \)

\( x < y \): \( x \) is less than \( y \)

\( x \geq y \): \( x \) is greater than or equal to \( x \)

\( x \leq y \): \( x \) is less than or equal to \( y \)

\( x == y \): \( x \) equals \( y \)

Q: What is the difference between == and = ?
A: == is for equality
   = is for assignment
Negation operator

“!” flips true/false

if (x >= y) {
    ...
}
Negation operator

“!” flips true/false

if \( x \geq y \) {
  ...
}

is equivalent to

if \( !(x < y) \) {
  ...
}

Boolean operators

&&: logical AND
||: logical OR

if (x > 6) {
    if (x < 9) {
        ...
    }
}
}
Boolean operators

&&: logical AND
||: logical OR

if (x > 6) {
  if (x < 9) {
    ...
  }
}

is equivalent to

if (x > 6 && x < 9) {
  ...
}
Increment and decrement operators

\( i = i + 1 \)

++i : prefix increment

i++ : postfix increment

\( i = i - 1 \)

--i : prefix decrement

i-- : postfix decrement
Increment and decrement operators

\[ i = i + 1 \]
\[ ++i : \text{prefix increment} \]

\[ i++ : \text{postfix increment} \]

\[ i = i - 1 \]
\[ --i : \text{prefix decrement} \]

\[ i-- : \text{postfix decrement} \]

● On their own the post and prefix forms behave the same.
● When part of an expression, however, they have different behavior.
Increment and decrement operators

```java
int output = 0;
int k = 10;

output = k++; // Assigns k to output, then increments k
System.out.println("k: " + k); // k: 11
System.out.println("output: " + output); // output: 10

output = 0;
int j = 10;

output = ++j; // Increments j, then assigns to output
System.out.println("j: " + j); // j: 11
System.out.println("output: " + output); // output: 11
```
Increment and decrement operators

```c
int output = 0;
int k = 10;
int j = 10;

k++ // Assigns k to output, then increments k
output = k++;  
k: 11
output: 10

++j // Increments j, then assigns to output
output = ++j;  
j: 11
output: 11
```
for loops

```java
for (initialization; condition; update) {
    statements
}
```
for loops

```java
for (int i = 0; i < 21; i++) {
    System.out.println("Floor number: " + i);
}
/* Output:
   Floor number: 0
   Floor number: 1
   Floor number: 2
   Floor number: 3
   ...
   Floor number: 18
   Floor number: 19
   Floor number: 20
*/
```
Branching statements

`continue` skips the current iteration of a loop and proceeds directly to the next iteration
Branching statements

`continue` skips the current iteration of a loop and proceeds directly to the next iteration

```java
for (int i = 0; i < 21; i++) {
  if (i == 13) continue;
  System.out.println("Floor number: " + i);
}
```

// What's the output?
Branching statements

*continue* skips the current iteration of a loop and proceeds directly to the next iteration

```java
for (int i = 0; i < 21; i++) {
    if (i == 13) continue;
    System.out.println("Floor number: "+ i);
}
```

/* Output
... 
Floor number: 12  
Floor number: 14  
... 
*/
Branching statements

*break* terminates a *for* or *while* loop

```java
for (int i = 1; i < 100; i++) {
    if (i == 82) {
        break;
    }
    System.out.println("Straw no. " + i);
}
```

// What's the output?
Branching statements

`break` terminates a for or while loop

```java
for (int i = 1; i < 100; i++) {
    if (i == 82)
        break;
    System.out.println("Straw no. " + i);
}
```

/* Output
Straw no. 1
Straw no. 2
...
Straw no. 81
*/
A loop within a loop

```java
for (int min = 0; min < 60; min++) {
    for (int sec = 0; sec < 60; sec++) {
        System.out.println(min + " min, " + sec + " sec");
    }
}
```
A loop within a loop

```java
for (int min = 0; min < 60; min++) {
    for (int sec = 0; sec < 60; sec++) {
        System.out.println(min + " min, " + sec + " sec");
    }
}
/* Output
  0 min, 0 sec
  0 min, 1 sec
  0 min, 2 sec
  ...
  59 min, 58 sec
  59 min, 59 sec
*/
```
Arrays
Arrays

An array is an indexed list of values.

You can make an array of any type int, double, String, etc..

All elements of array must have the same type.
Arrays Example

values

| 0 | 1 | 2 | 3 | ... | n-1 |

indexes
Arrays Example

double[] arr;

<table>
<thead>
<tr>
<th>values</th>
<th>5.0</th>
<th>2.4</th>
<th>11.9</th>
<th>-22.0</th>
<th>...</th>
<th>2.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>indexes</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>...</td>
<td>n-1</td>
</tr>
</tbody>
</table>
Defining Arrays

An array is defined using TYPE [ ]

\texttt{int[]} \ values;  // array of int

Arrays are just another type.

\texttt{int[][]} \ values;  // \texttt{int[]} is a type
\texttt{// (int[])[] values;}
Creating Arrays

To create an array of a given size, use operator `new`:

```c
int[] values = new int[5];
```

or you may use a variable to specify the size:

```c
int size = 12;
int[] values = new int[size];
```
Curly braces can be used to initialize an array. It can **ONLY** be used when you declare the variable.

```java
int[] values = { 12, 24, -23, 47 };```

**Initializing Arrays**
Quiz time!

Is there an error in this code?

```c
int[] values;
values = {1, 2.5, 3, 3.5, 4};
```
Accessing Arrays (1)

To access elements of array, use the \[ \] operator:

\[
\text{int}\[\] \text{values} = \{ 0, 5, 10, 15 \};
\]

\[
\text{values}[3] = 18;
\]
To access elements of array, use the \[\] operator:

```java
int[] values = {0, 5, 10, 15};
values[3] = 18;
int x = values[1] + 3;
```
Accessing Arrays (2)

The index starts at zero and ends at length-1.

```java
int[] values = new int[3];
values[0] = 12;
```
Accessing Arrays (2)

The index starts at zero and ends at length-1.

```
int[] values = new int[3];
values[0] = 12; // CORRECT
```
Accessing Arrays (2)

The index starts at zero and ends at length-1.

```java
int[] values = new int[3];
values[0] = 12; // CORRECT
values[2] = 12; // CORRECT
```
Accessing Arrays (2)

The index starts at zero and ends at length-1.

```
int[] values = new int[3];
values[0] = 12;  // CORRECT
values[2] = 12;  // CORRECT
values[3] = 12;  // WRONG!! compiles but throws an Exception at run-time
```

```java
0 1 2
12 12 ?
```
Accessing Arrays (2)

The index starts at zero and ends at length-1.

values[3] = 12; // WRONG!!

Called an **out of bounds access** error. Java differs from the original C standard in that it will throw an error here, preventing **buffer overflows**, or writing to adjacent memory and causing security concerns.
The *length* variable

Each array has a *length* variable built-in that contains the length of the array.

```java
int[] values = new int[12];
int size = values.length; // 12

int[] values2 = {1,2,3,4,5}
int size2 = values2.length; // 5
```
String arrays

A side note

```java
public static void main (String[] args){
    System.out.println(args.length);
    System.out.println(args[0]);
    System.out.println(args[1]);
}
```
Combining Loops and Arrays
Print square of elements in values

```cpp
int[] values = {1,2,3,5,7};
```
Array looping using *for*

```java
int[] values = {1,2,3,5,7};
int square = 0;

for (int i=0;  i < values.length; i++) {
    square = values[i] * values[i];
    System.out.println(square);
}
```
int[] values = {1,2,3,5,7};
int i = 0;
int square = 0;
while (i < values.length) {
    square = values[i] * values[i];
    System.out.println(square);
    i++;
}
Enhanced *for* loop

Feature in J2SE 5.0 to iterate through values in an array

```java
for (int i : values) { // for each int in values
   System.out.println(i);
}
```

The same as:

```java
for (int i=0; i<values.length; i++) {
   System.out.println(values[i]);
}
```
Assignment 3

A group of friends participate in the Boston Marathon.

Find the best performer.

Find the second-best performer.
Methods

public static void NAME () {
    STATEMENTS
}

Methods

```java
public static void NAME () {
    STATEMENTS
}
```

To call a method:

```java
NAME ();
```
class Methods {
    public static void newLine () {
        System.out.println("\n");
    }

    public static void main (String[] arguments) {
        System.out.println("Above 3 lines");
        newLine();
        newLine();
        newLine();
        System.out.println("Below 3 lines");
    }
}
class Methods {
    public static void newLine () {
        System.out.println('');
    }

    public static void threeNewLines() {
        newLine(); newLine(); newLine();
    }

    public static void main (String[] arguments) {
        System.out.println("Above 3 lines");
        threeNewLines();
        System.out.println("Below 3 lines");
    }
}
Parameters

public static void NAME (TYPE NAME) {
   STATEMENTS
}

To call a method:

NAME (EXPRESSION) ;
class Methods {
    public static void newLine () {
        System.out.println("\n");
    }

    public static void newLine (int i) { // same name, diff params
        for (int j = 0; j < i; j++) newLine();
    }

    public static void main (String[] arguments) {
        int numLines = 3;
        newLine (numLines);
    }
}
```java
class Square {
    public static void printSq (int x) {
        System.out.println(x*x);
    }
    public static void main (String[] arguments) {
        int value = 3;
        printSq(value);
        printSq(4);
        printSq(value + 2);
    }
}
// Output: 9
    16
    25
```
class Square {
    public static void printSq (int x) {
        System.out.println(x*x);
    }
    public static void main (String[] arguments) {
        int value = 3;
        printSq("square me");
        printSq(5.5);
        printSq("10");
    }
}

// What's wrong?
Multiple Parameters

public static void NAME (TYPE NAME, TYPE NAME) {
    STATEMENTS
}

To call:

NAME (arg1, arg2);
class Multiply {
    public static void times (double a, double b) {
        System.out.println(a * b);
    }

    public static void main (String[] arguments) {
        times(2, 2);
        times(1.5, 10);
    }
}
public static TYPES NAME () {
    STATEMENTS
    return EXPRESSION;
}

void means "no type"
class Square {
    public static double square (double x) {
        return (x*x);
    }
    public static void main (String[] arguments) {
        System.out.println("4 squared is " + square(4));
        double cubed = 4 * square(4);
        System.out.println("4 cubed is " + cubed);
    }
}
```java
class Square {
    public static double square (double x) {
        return x*x;
    }

    public static void main (String[] arguments) {
        System.out.println("4 squared is " + square(4));
        double cubed = 4 * square(4);
        System.out.println("4 cubed is " + cubed);
    }
}

// Output: 4 squared is 16.0
4 cubed is 64.0
```
Introduction to OOP

- OOP stands for Object-Oriented Programming

- Distinct from procedural programming, which emphasizes discrete reusable code blocks that take in a value, perform some action and return the value

- OOP takes the model of interacting Objects, with their own states and behaviors
Examples of an Object

**Object**: Smart Phone

**States**: brand, model, on, off, hibernating, volume level, screen brightness, on phone call

**Behaviors**: change volume, change brightness, call contact, check email, send text,
Smart Phone

class SmartPhone {
    String brand = "Samsung";
    String model = "Nexus S";
    int volumeLevel = 3; // range of 7
    boolean onCall = false;

    void raiseVolume (){
        if (volumeLevel < 7) volumeLevel++;
    }
}

Attributes

Methods
Many instances of Smart Phone objects

brand = "Samsung"       brand = "Acer"          brand = "HTC"
raiseVolume()                            ...                                ...                                ...
onCall()                                                ...                                ...
sendText(String m, int num)                ...                                ...

Benefits of OOP

- Models real and abstract entities more realistically by grouping together their states and behaviors.

- *Abstraction*: We only need to know what the object can do (its methods) and its states (its values) not how these are implemented.

- *Encapsulation*: We hide the internals of objects that should remain stable.