Notes: User Input and Conditionals

Scanner class / User input

- We will be using `Scanner` for user input
- In order to use `Scanner` you need to add an import statement to the top of your code: `import java.util.*;`
- In main, you will use `Scanner console = new Scanner(System.in);` to create a Scanner object named console that you can pass to any of your methods that need user input
- Note: The `console` name is arbitrary, if it makes sense use a different name
- Note: You should only ever construct 1 Scanner object and pass it in as a parameter to only the methods that need it
- **token**: A sequence of characters that are not white space (e.g., tabs, spaces, etc)

Scanner methods

Method	**Description**
`nextInt()`	reads a token of user input as an `int`; can only read ints, otherwise error
`nextDouble()`	reads a token of user input as a `double`; can read doubles and ints (converts to double)
`next()`	reads a token of user input as a `String`
`nextLine()`	reads a line of user input as a `String`; will include white space characters

Example

```
import java.util.Scanner;
public class UserInputExample {
   public static void main(String[] args) {
      Scanner console = new Scanner(System.in);
      age(console);
      int diff = age(15);
      System.out.println(diff + " years until you are 40.");
   }

   // this method PROMPTS for an age and PRINTS the result
   public static void age(Scanner console) {
      System.out.print("How old are you? ");
      int age = console.nextInt();
      System.out.println("You'll be 40 in " + (40 - age) + " years.");
   }

   // this method TAKES an age and RETURNS the result
   public static int age(int age) {
      return 40 - age;
   }
}
```

Expressions that result in a boolean (true/false)

Relational Operators

Operator	**Description**	**Example**	**Result**
==	equals (for primitive types)	1 + 1 == 2	true
s.equals()	equals (for Strings and other reference types)	s.equals("hi")	
!=	does not equal (for primitive types)	3.2 != 2.5	true
!s.equals()	not equals (for Strings and other reference types)	!s.equals("hi")	
<	less than	10 < 5	false
>	greater than	10 > 5	true
<=	less than or equal to	126 <= 100	false
>=	greater than or equal to	5.0 >= 5.0	true

Logical operators

Operator	**Description**	**Example**	**Result**
&&	and	(2 == 3) && (-1 < 5)	false
||	or	(2 == 3) || (-1 < 5)	true
!	not	!(2 == 3)	true

- && (and) is used in Java to check if two conditions are BOTH true
- || (or) is used in Java to check if AT LEAST ONE of two conditions is true
- ! (not, sometimes read as "bang") is used in Java to negate a condition (make true become false, or make false become true).

Logical Truth Table

p	**q**	**p && q**	**p || q**
true	true	true	true
true	false	false	true
false	true	false	true
false	false	false	false

Negating a boolean

p	**!p**
true	false
false	true

Conditionals

- else can only be used when paired with an if
- Note: there should NOT be a semicolon at the end of an if-statement condition
- In Java, indentation does not cause statements to belong together. You must use {}s

if statements in sequence

```
// independent tests; not exclusive
// 0, 1, or many of the statement(s) may execute
// every test in every if block is checked
if (test) {
   statement(s);
}
if (test) {
   statement(s);
}
if (test) {
   statement(s);
}
```

if / else if (no else)

```
// 0, or 1 of the if blocks may execute
// at most only 1 of the if blocks execute
// it could be the case that 0 if blocks execute because there is no else
if (test) {
   statement(s);
} else if (test) {
   statement(s);
} else if (test) {
   statement(s);
}
```

if / else if / else

```
// exactly 1 of the if blocks will execute
if (test) {
   statement(s);
} else if (test) {
   statement(s);
} else {
   statement(s);
}
```

- If statement conditions are evaluated in sequence (top to bottom). If the condition is `true`, then the associated block is executed and the rest of the conditions are skipped. If the condition is `false`, the next condition is tested.
- If there is no `else`, then it is possible that none of the blocks are executed (if none of the conditions were true). However, if there is an `else`, then if the `else` is reached (meaning all conditions before it were false) its associated block will be executed (as `else` basically means "otherwise do this")

Notes: Common Algorithms and printf

Common Algorithms

These are common patterns in programming that are important to know!

Cumulative Sum

```
// returns the sum of integers from 1 up to n
public static int calculateSum(int n) {
  int sum = 0;
  for (int i = 1; i <= n; i++) {
    sum = sum + i;
  }
  return sum;
}
```

Max

```
public static int findMax(Scanner console, int n) {
    int max = Integer.MIN_VALUE;

    for (int i = 0; i < n; i++) {
        System.out.print("Enter a value: ");
        int num = console.nextInt();

        if (num > max) {
            max = num;
        }
    }
    return max;
}
```

Even or Odd

```
public static void evenOrOdd(int n) {
    if (n % 2 == 0) {
        System.out.println(n + " is even.");
    } else {
        System.out.println(n + " is odd.");
    }
}
```

Replicate Entire String

```
// returns a String containing n replications of s
public static String replicate(String s, int n) {
    String output = "";
    for (int i = 0; i < n; i++) {
      output = output + s;
    }
    return output;
}
```

Reverse String

```
public static String reverse(String phrase) {
    String output = "";
    for (int i = 0; i < phrase.length(); i++) {
        output = phrase.charAt(i) + output;
    }
    return output;
}
```

Using printf

- The f in printf stands for formatted
- Allows you to format what you are printing

Example

```
double x = 38.421;
double y = 152.734009;
// the below line will output: formatted numbers: 38.42, 152.7
System.out.printf("formatted numbers: %.2f, %.1f\n", x, y);
```

Common Format Specifiers

Specifier	**Result**
%.2f	Floating-point number, rounded to nearest hundredth
%d	Integer
%6d	Integer, left-aligned, 6-space-wide field
%f	Floating-point number
%16.3f	Floating-point number, rounded to nearest thousandth, 16-space-wide field
%s	String