Lecture 04

Math functions, Chars, and Strings
Math Class

• Talked about it a bit with Math.random()

• Built-in to Java.lang, so don’t have to import it.

• Static, so we don’t need an instance of it.

• What other methods and constants are there?

  • To the Javadoc! (https://docs.oracle.com/javase/7/docs/api/)
Class Stuff

• New tutoring calendar link: HERE

• Link to even-numbered solutions from book: HERE
Trig Methods

- sin(radians) -> Sine of an angle in radians
- cos(radians) -> Cosine of an angle in radians
- tan(radians) -> Tangent of an angle in radians
- toRadians(degrees) -> Angle in radians of angle in degrees
- toDegree(radians) -> Angle in degrees of angle in radians
- asin(a) -> angle in radians for the inverse of sine
- acos(a) -> angle in radians for the inverse of cosine
- atan(a) -> angle in radians for the inverse of tangent
Radians and Degrees

• 1 degree = $\pi / 180$ radians

• 90 degrees = $\pi/2$ radians
Note: This is not a geometry class! I'll give you formulas if we need them, you won't have to figure them out on your own.
Exponent Methods

- \( \text{exp}(x) \rightarrow e^x \)
- \( \text{log}(x) \rightarrow \text{natural log of } x \)
- \( \text{log10}(x) \rightarrow \text{base 10 log of } x \)
- \( \text{pow}(a, b) \rightarrow a^b \)
- \( \text{sqrt}(x) \rightarrow \text{square root of } x \)
Rounding Methods

- ceil(x) -> rounded UP to a double equal to nearest int
- floor(x) -> rounded DOWN to a double equal to nearest int
- rint(x) -> rounded to a double equal to nearest int. if right in the middle, rounds to even int
- round(x) -> rounded to nearest int. (rounds up from .5)
public class RoundingMethods {

    public static void main(String[] args) {
        double num = -2.1;

        System.out.println(Math.ceil(num));
        System.out.println(Math.floor(num));
        System.out.println(Math.rint(num));
        System.out.println(Math.round(num));
    }
}
• min and max of 2 numbers

• ex. Math.max(2,3)

• Math.abs(-2.1)
min, max, abs

When would you use these?

• min and max are useful when sorting

• abs can be used for getting distance between points
Chars

- One single character

- Character literal in single quotation marks. (if you use "", it's not a char anymore, it's a String)

  - ex. char letter = 'B';

  - ex. char nextChar = '7';
Chars

• Java supports Unicode, so most characters should work just fine

• chars also have ASCII values
Chars

- Java supports Unicode, so most characters should work just fine
- chars also have ASCII values that can be used by casting to and from ints

<table>
<thead>
<tr>
<th>Characters</th>
<th>Code Value in Decimal</th>
<th>Unicode Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>'0' to '9'</td>
<td>48 to 57</td>
<td>\u0030 to \u0039</td>
</tr>
<tr>
<td>'A' to 'Z'</td>
<td>65 to 90</td>
<td>\u0041 to \u005A</td>
</tr>
<tr>
<td>'a' to 'z'</td>
<td>97 to 122</td>
<td>\u0061 to \u007A</td>
</tr>
</tbody>
</table>
public class CharsAndNumbers {
    public static void main(String[] args) {

        char myChar = 'A';
        int fiveAfter = myChar + 5;

        System.out.println((int) myChar);
        System.out.println((char) fiveAfter);
    }
}
Chars

• How could you generate a random uppercase letter?
int randomNumber = 65 + (int) (Math.random() * 26);
System.out.println((char) randomNumber);
Special Characters: Escape sequences

• Super important when parsing data

• It's how you put special characters into Strings

• Names come from old printer commands

• even though they are technically two characters, they are one “Char” each
Escape sequences

- `\b`  -> Backspace
- `\t`  -> Tab
- `\n`  -> Linefeed
- `\f`  -> Formfeed
- `\r`  -> Carriage Return
- `\\` -> Backslash
- `\"` -> Double quote
public class SpecialChars {
    public static void main(String[] args) {
        System.out.println("Tab \t example");
        System.out.println("Linefeed \n example");
        System.out.println("Backslash \ \ example");
        System.out.println("Double quote \" example");
    }
}
\ BACKSLASH
\\ REAL BACKSLASH
\\\ REAL REAL BACKSLASH
\\\\ ACTUAL BACKSLASH, FOR REAL THIS TIME
\\\\\ ELDER BACKSLASH
\\\\\\ BACKSLASH WHICH ESCAPES THE SCREEN AND ENTERS YOUR BRAIN
\\\\\\\ BACKSLASH SO REAL IT TRANSCENDS TIME AND SPACE
\\\\\\\\ BACKSLASH TO END ALL OTHER TEXT
\\\\\\\\... THE TRUE NAME OF BA'AL, THE SOUL-EATER
Character Class

• Java has some helpful methods that you can use for chars built into the **Character** class

• These are used by calling, for example: `Character.isDigit('a')`

• **Note:** unlike strings, you use the static class `Character` (like `Math`) and not an instance of a char object
Helpful methods in Character class

- `isDigit(ch)` -> is true if a number
- `isLetter(ch)` -> is true if a letter
- `isLetterOrDigit(ch)` -> is true if letter or digit (not escape character)
- `isLowerCase(ch)` -> is true if lowercase letter
- `isUpperCase(ch)` -> is true if uppercase letter
- `toLowerCase(ch)` -> returns the lowercase of the character
- `toUpperCase(ch)` -> returns the uppercase of the character
Helpful methods in Character class

Character.isDigit('a')
Strings

• Used to represent a string of characters

• Syntax:
  
  • String exampleString = “Hey, this thing is a String”;

• Not a primitive type (like int or double), but reference type.

• That also means String has a capital S like other classes that are not primitive data types.
Useful methods for Strings

- `length()` • number of characters in a string
- `charAt(index)` • returns the character at the point asked
- `concat(s1)` • concatenates this string with s1
- `toUpperCase()` • returns string, but all letters uppercase
- `toLowerCase()` • returns string, but all letters uppercase
- `trim()` • returns a new string with no whitespace in front or behind
instance methods vs. static methods

• Strings are objects, so the methods on the last slide can only be called from a specific instance of the object, not on a static object.

• What’s an example of another object we’ve used to call instance methods already?

• What’s an example of an object we’ve used to call static methods already?
Get first and last character of a string
Get first and last character of a string

```java
public class GetCharactersFromString {

    public static void main(String[] args) {
        char firstChar;
        char lastChar;
        String exampleString = "Hey, this thing is a String";

        firstChar = exampleString.charAt(0);
        lastChar = exampleString.charAt(exampleString.length()-1);

        System.out.println("The first character is " + firstChar + " and the last is " + lastChar);
    }
}
```
0-index for strings

```
message.charAt(0)  message.length() is 15  message.charAt(14)
```
Reading in Strings and Chars

• Use Scanner object to read in strings using `next()` and `nextLine()` methods

• `next()` reads until the next whitespace character (a space, newline char, tab, etc)

• `nextLine()` reads the entire line of text

• to get a char, use `nextLine()`, then `charAt(0)` on the string
Reading in Strings and chars

• **Note:** Do not try to use `nextLine()` after `next()` or `nextInt()`, `nextDouble()`, etc. It will fail.

• In order to use `nextLine()` after one of the other methods, add in a `nextLine()` call that is not assigned to anything. For example:
  ```java
  input.nextLine()
  ```
Reading in Strings and chars

```java
import java.util.*;
public class ReadingInStrings {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);

        String s1, s2, s3;
        String s4;

        System.out.println("Please enter 3 words, separated by spaces: ");
        s1 = input.next();
        s2 = input.next();
        s3 = input.next();

        System.out.println(s1);
        System.out.println(s2);
        System.out.println(s3);

        // Needed to use nextLine below after using next() or nextInt(), etc
        input.nextLine();

        System.out.print("Please enter a sentence: ");
        s4 = input.nextLine();

        System.out.println(s4);
    }
}
```
import java.util.*;

public class GuessAWord {

    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        String secretWord = "secret";
        String userGuess;

        System.out.println("Try to guess the word I'm thinking");
        userGuess = input.nextLine();

        if (secretWord == userGuess){
            System.out.println("You guessed it!");
        } else {
            System.out.println("Sorry, that's not it.");
        }
    }
}
Comparing Strings

- `equals(s1)`
- `equalsIgnoreCase(s1)`
- `compareTo(s1)`
- `compareToIgnoreCase(s1)`
- `startsWith(prefix)`
- `endsWith(suffix)`
- `contains(s1)`

- Returns true if string is equal to `s1`
- Same as above, but case insensitive
- Compare strings and return an int >0, 0, or <0
- Same as above, but case insensitive
- Returns true if string starts with prefix
- Returns true if string ends with suffix
- Returns true if `s1` in inside string
import java.util.*;

public class GuessAWord {

    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        String secretWord = "secret";
        String userGuess;

        System.out.println("Try to guess the word I'm thinking");
        userGuess = input.nextLine();

        if (secretWord.equals(userGuess)){
            System.out.println("You guessed it!");
        } else {
            System.out.println("Sorry, that's not it.");
        }
    }
}

Testing if strings are the same

• Don’t use == to compare strings! == only checks to see if the strings refer to the same object, not whether or not they have the same contents

• use string1.equals(string2)
Substrings

• `substring(beginIndex)`  -> Returns section of the string starting at `beginIndex`, to the end of the string

• `substring(beginIndex, endIndex)`  -> Returns section of the string starting at `beginIndex`, to `endIndex - 1`
public class GetSubstring {
    public static void main(String[] args) {
        String exampleString = "Hey, this thing is a String";
        String sub;

        sub = exampleString.substring(21);
        System.out.println(sub);

        sub = exampleString.substring(10, 15);
        System.out.println(sub);
    }
}
Searching Strings

• `indexOf(ch), indexOf(s)`

• `indexOf(ch, fromIndex), indexOf(s, fromIndex)`

• `lastIndexOf(ch), lastIndexOf(s)`

• `lastIndexOf(ch, fromIndex), lastIndexOf(s, fromIndex)`
public class searchString {
    public static void main(String[] args) {
        String exampleString = "Hey, this thing is a String";

        System.out.println(exampleString.indexOf("this"));
        // System.out.println(exampleString.indexOf("th");
        // System.out.println(exampleString.indexOf("a");
        // System.out.println(exampleString.indexOf("z");
    }
}
Strings to numbers, numbers to strings

```java
int numInt = Integer.parseInt(intString);
double numDouble = Double.parseDouble(doubleString);

String s = num + "";
```
Challenge: Small data parsing at AMNH
Challenge: Small data parsing at AMNH

- For an interactive media piece in an exhibit last year, I needed to parse a list of foods in the following format:
  - [G,B]-[FoodName]
  - ex. G-Broccoli, B-Chips
Challenge: Small data parsing at AMNH

• Write a program that will parse one line of that file:
  • G-Carrots

• And report to the user what the food was, and whether it’s a good food, or a bad food
public class ParseFood {
    public static void main(String[] args) {
        String foodText = "G-Carrots";
        int dashIndex;
        String goodOrBad;
        String foodName;

        dashIndex = foodText.indexOf('-');
        goodOrBad = foodText.substring(0, dashIndex);

        System.out.print(foodText.substring(dashIndex+1) + " are ");
        if (goodOrBad.equals("G")){
            System.out.println("good food!");
        } else {
            System.out.println("bad food!");
        }
    }
}