Math

Based on slides by Omar Ibrahim
https://courses.cs.washington.edu/courses/cse142/21su/

Java's `Math` class

https://docs.oracle.com/javase/8/docs/api/java/lang/Math.html

Method name	Description
`Math.abs(`*value*`)`	absolute value
`Math.ceil(`*value*`)`	rounds up
`Math.floor(`*value*`)`	rounds down
`Math.log10(`*value*`)`	logarithm, base 10
`Math.max(`*value1*`, `*value2*`)`	larger of two values
`Math.min(`*value1*`, `*value2*`)`	smaller of two values
`Math.pow(`*base*`, `*exp*`)`	*base* to the *exp* power
`Math.random()`	random `double` between 0 and 1
`Math.round(`*value*`)`	nearest whole number
`Math.sqrt(`*value*`)`	square root

`Math.sin(`*value*`)`
`Math.cos(`*value*`)`
`Math.tan(`*value*`)`
	sine/cosine/tangent of an angle in radians

`Math.toDegrees(`*value*`)` `Math.toRadians(`*value*`)`	convert degrees to radians and back

Constant	Description
`Math.E`	2.7182818...
`Math.PI`	3.1415926...

`Math` questions

- Evaluate the following expressions:
  - `Math.abs(-1.23)`
  - `Math.pow(3, 2)`
  - `Math.pow(10, -2)`
  - `Math.sqrt(121.0) - Math.sqrt(256.0)`
  - `Math.round(Math.PI) + Math.round(Math.E)`
  - `Math.ceil(6.022) + Math.floor(15.9994)`
  - `Math.abs(Math.min(-3, -5))`

- `Math.max` and `Math.min` can be used to bound numbers. Consider an `int` variable named `age`.
  - What statement would replace negative ages with 0?
  - What statement would cap the maximum age to 40?

Incompatible types

- Some `Math` methods return `double` or other non-`int` types.

```
int x = Math.pow(10, 3); // ERROR: incompatible types
```

- What if you wanted to store a `double` in an `int` variable? (maybe you don't care about the decimal part)

Type casting

- **type cast**: A conversion from one type to another.
  - To promote an `int` into a `double` to get exact division from `/`
  - To truncate a `double` from a real number to an integer

- Syntax:

  (**type**) **expression**

  Examples:

```
double result = (double)19 / 5;      // 3.8
int result2 = (int)result;           // 3
int x = (int)Math.pow(10, 3);        // 1000
```

More about type casting

- Type casting has high precedence and only casts the item immediately next to it.
  - `double x = (double) 1 + 1 / 2;  // 1.0`
  - `double y = 1 + (double) 1 / 2;  // 1.5`
- You can use parentheses to force evaluation order.
  - `double average = (double)(a + b + c) / 3;`
- A conversion to `double` can be achieved in other ways.
  - `double average = 1.0 * (a + b + c) / 3;`

Easy trick for rounding

- If we want to round a double to the nearest int, that's easy:
  - `double myNum = 3.64716351;`
  - `int roundedNum = Math.round(myNum);  // 4`
  - `int truncatedNum = (int) myNum;  // 3`
- What if we want to round a double to, say, 2 decimal places?
  - There are many ways.
    Some helpful packages like DecimalFormat or BigDouble
  - Quick Way:
    - `double shortDouble = Math.round(myNum * 100.0) / 100.0;`
  - Change 100.0 depending how many decimal places you want.

Exercise

- In physics, the *displacement* of a moving body represents its change in position over time while accelerating.
  - Given initial velocity v_0 in m/s, acceleration a in m/s^2, and elapsed time t in s, the displacement of the body is:
  - Displacement = $v_0 t + \frac{1}{2} a t^2$

- Write a method `displacement` that accepts v_0, a, and t and computes and returns the change in position.
  - Example: `displacement(3.0, 4.0, 5.0)` returns `65.0`

Exercise solution

```
public static double displacement(double v0, double a, double t) {
    double d = v0 * t + 0.5 * a * Math.pow(t, 2);
    return d;
}
```

Strings

Strings

- **string**: An object storing a sequence of text characters.
  - Unlike most other objects, a `String` is not created with `new`.

```
String name = "text";
String name = expression (with String value);
```

  - Examples:

```
String names = "Alice and Bob";
int x = 3;
int y = 5;
String point = "(" + x + ", " + y + ")";
```

Indexes

- Characters of a string are numbered with 0-based *indexes*:

```
String name = "M. Mouse";
```

index	0	1	2	3	4	5	6	7
character	M	.		M	o	u	s	e

  - First character's index : 0
  - Last character's index : 1 less than the string's length

  - The individual characters are values of type `char` (seen later)

`String` methods

https://docs.oracle.com/javase/8/docs/api/java/lang/String.html

Method name	Description
`indexOf(`**str**`)`	index where the start of the given string appears in this string (-1 if not found)
`length()`	number of characters in this string
`substring(`**index1**`, `**index2**`)` or `substring(`**index1**`)`	the characters in this string from *index1* (inclusive) to *index2* (exclusive); if *index2* is omitted, grabs till end of string
`toLowerCase()`	a new string with all lowercase letters
`toUpperCase()`	a new string with all uppercase letters

- These methods are called using the dot notation:

```
String starz = "Prince vs. Michael";
System.out.println(starz.length());   // 18
```

String method examples

```
// index      012345678901
String s1 = "Stuart Reges";
String s2 = "Marty Stepp";

System.out.println(s1.length());          // 12
System.out.println(s1.indexOf("e"));      // 8
System.out.println(s1.substring(7, 10)); // "Reg"

String s3 = s2.substring(1, 7);
System.out.println(s3.toLowerCase());     // "arty s"
```

- Given the following string:

```
// index        01234567890123456789 01
String book = "Building Java Programs";
```

  - How would you extract the word "Java" ?

Modifying strings

- Methods like `substring` and `toLowerCase` build and return a new string, rather than modifying the current string.

```
String s = "Mumford & Sons";
s.toUpperCase();
System.out.println(s);    // Mumford & Sons
```

- To modify a String variable's value, you must reassign it:

```
String s = "Mumford & Sons";
s = s.toUpperCase();
System.out.println(s);    // MUMFORD & SONS
```

Exercise: Name border

```
HELENE
HELEN
HELE
HEL
HE
H
HE
HEL
HELE
HELEN
HELENE
MARTIN
MARTI
MART
MAR
MA
M
MA
MAR
MART
MARTI
MARTIN
```

- Prompt the user for full name
- Draw out the pattern to the left

Comparing strings

- Relational operators such as < and == fail on objects.

```
Scanner console = new Scanner(System.in);
System.out.print("What is your name? ");
String name = console.next();
if (name == "Barney") {
    System.out.println("I love you, you love me,");
    System.out.println("We're a happy family!");
}
```

  - This code will compile, but it will not print the song.

  - == compares objects by *references* (seen later), so it often gives `false` even when two `String`s have the same letters.

The `equals` method

- Objects are compared using a method named `equals`.

```
Scanner console = new Scanner(System.in);
System.out.print("What is your name? ");
String name = console.next();
if (name.equals("Barney")) {
    System.out.println("I love you, you love me,");
    System.out.println("We're a happy family!");
}
```

- Technically this is a method that returns a value of type `boolean`, the type used in logical tests.

`String` test methods

Method	Description
`equals(`**str**`)`	whether two strings contain the same characters
`equalsIgnoreCase(`**str**`)`	whether two strings contain the same characters, ignoring upper vs. lower case
`startsWith(`**str**`)`	whether one contains other's characters at start
`endsWith(`**str**`)`	whether one contains other's characters at end
`contains(`**str**`)`	whether the given string is found within this one

```
String name = console.nextLine();
if (name.startsWith("Dr.")) {
    System.out.println("Will you marry me?");
} else if (name.equalsIgnoreCase("buTteRs")) {
    System.out.println("You're grounded, young man!");
}
```

String documentation: http://docs.oracle.com/javase/7/docs/api/java/lang/String.html

Strings question

- Write a program that reads two people's first names and suggests a name for their child.
  - The suggestion is the concatentation of the first halves of both names.

Example Output:

```
Parent 1 first name? Danielle
Parent 2 first name? John
Suggested baby name: DANIJO
```

Strings answer

```
// Suggests a baby name based on parents' names.
import java.util.*;

public class BabyNamer {
   public static void main(String[] args) {
      Scanner s = new Scanner(System.in);
      System.out.print("Parent 1 first name? ");
      String name1 = s.next();
      System.out.print("Parent 2 first name? ");
      String name2 = s.next();

      System.out.println("Suggested name: " +
                         suggestChildName(name1, name2).toUpperCase());
   }

   ...
```

Strings answer (cont.)

```
…  // Danielle    John

// Suggests a child's name for parents with the given names.
public static String suggestChildName(String name1, String name2) {
   String halfName1 = getHalfName(name1);
   String halfName2 = getHalfName(name2);
   String name;
      name = halfName1 + halfName2;
   return name;
}

// Return the first half of the given name.
public static String getHalfName(String name) {
   int halfIndex = name.length() / 2;
   return name.substring(0, halfIndex);
}

}
```

char

Type `char`

- **`char`** : A primitive type representing single characters.
  - A `String` is stored internally as an array of `char`

```
String s = "nachos";
```

index	*0*	*1*	*2*	*3*	*4*	*5*
value	'n'	'a'	'c'	'h'	'o'	's'

  - It is legal to have variables, parameters, returns of type `char`
    - surrounded with apostrophes: `'a'` or `'4'` or `'\n'` or `'\''`

```
char initial = 'J';
System.out.println(initial);              // J
System.out.println(initial + " Joyce");   // J Joyce
```

The `charAt` method

- The `char`s in a `String` can be accessed using the `charAt` method.
  - accepts an `int` index parameter and returns the `char` at that index

```
String food = "cookie";
char firstLetter = food.charAt(0);    // 'c'

System.out.println(firstLetter + " is for " + food);
```

- You can use a `for` loop to print or examine each character.

```
String major = "CSE";
for (int i = 0; i < major.length(); i++) {     // output:
    char c = major.charAt(i);                  // C
    System.out.println(c);                     // S
}                                              // E
```

char vs. String

- "h" is a String, but 'h' is a char (they are different)
- A String is an object; it contains methods.

```
String s = "h";
s = s.toUpperCase();          // "H"
int len = s.length();         //  1
char first = s.charAt(0);     // 'H'
```

- A char is primitive; you can't call methods on it.

```
char c = 'h';
c = c.toUpperCase();              // ERROR
s = s.charAt(0).toUpperCase();    // ERROR
```

`char` vs. `int`

- Each `char` is mapped to an integer value internally
  - Called an **ASCII value**

`'A'` is 65	`'B'` is 66	`' '` is 32
`'a'` is 97	`'b'` is 98	`'*'` is 42

  - Doing "math" on a `char` causes automatic conversion to `int`.
    `'a' + 10` is 107, `'A' + 'A'` is 130
  - To convert an `int` into the equivalent `char`, type-cast it.
    `(char) ('a' + 2)` is `'c'`

(Optional) `printf`

Formatting text with `printf`

```
System.out.printf("format string", parameters);
```

- A format string can contain *placeholders* to insert parameters:
  - `%d` integer
  - `%f` real number
  - `%s` string
    - these placeholders are used instead of + concatenation
  - Example:

```
int x = 3;
int y = -17;
System.out.printf("x is %d and y is %d!\n", x, y);
                    // x is 3 and y is -17!
```

    - `printf` does not drop to the next line unless you write `\n`

`printf` width

- `%`**W**`d` integer, **W** characters wide, right-aligned
- `%-`**W**`d` integer, **W** characters wide, *left*-aligned
- `%`**W**`f` real number, **W** characters wide, right-aligned
- `...`

```
for (int i = 1; i <= 3; i++) {
    for (int j = 1; j <= 10; j++) {
        System.out.printf("%4d", (i * j));
    }
    System.out.println();   // to end the line
}
```

Output:

```
   1   2   3   4   5   6   7   8   9  10
   2   4   6   8  10  12  14  16  18  20
   3   6   9  12  15  18  21  24  27  30
```

`printf` precision

- `%.`**`D`**`f` real number, rounded to **D** digits after decimal
- `%`**`W`**`.`**`D`**`f` real number, **W** chars wide, **D** digits after decimal
- `%-`**`W`**`.`**`D`**`f` real number, **W** wide (left-align), **D** after decimal

```
double gpa = 3.253764;
System.out.printf("your GPA is %.1f\n", gpa);
System.out.printf("more precisely: %8.3f\n", gpa);
```

Output:

```
your GPA is 3.3
more precisely:    3.254
```

3

8

`printf` question

- Modify our `Receipt` program to better format its output.
  - Display results in the format below, with 2 digits after .

- Example log of execution:

```
How many people ate? 4
Person #1: How much did your dinner cost? 20.00
Person #2: How much did your dinner cost? 15
Person #3: How much did your dinner cost? 25.0
Person #4: How much did your dinner cost? 10.00

Subtotal:  $70.00
Tax:       $5.60
Tip:       $10.50
Total:     $86.10
```

printf answer (partial)

```
    ...

    // Calculates total owed, assuming 8% tax and 15% tip
    public static void results(double subtotal) {
        double tax = subtotal * .08;
        double tip = subtotal * .15;
        double total = subtotal + tax + tip;

        // System.out.println("Subtotal: $" + subtotal);
        // System.out.println("Tax: $" + tax);
        // System.out.println("Tip: $" + tip);
        // System.out.println("Total: $" + total);

        System.out.printf("Subtotal: $%.2f\n", subtotal);
        System.out.printf("Tax:      $%.2f\n", tax);
        System.out.printf("Tip:      $%.2f\n", tip);
        System.out.printf("Total:    $%.2f\n", total);
    }
}
```

Date and Time

Date and Time

- Java Tutorial: https://docs.oracle.com/javase/tutorial/datetime/iso/overview.html
  - import java.time.*; // import all of the time classes

- There are two basic ways to represent time. One way represents time in human terms, referred to as human time, such as year, month, day, hour, minute and second. The other way, machine time, measures time continuously with high precision.
- First determine what aspects of date and time you require, and then select the class for those needs. Do you want human time or machine time? Date and time? Date only? If you need a date, do you need month, day, and year, or a subset? Also, do you need to think about time zones?

Date and Time

- I will ignore time zones in class.
  If you are interested, then you can go read the tutorial ☺

- Human readable time, especially time zones, is very difficult.

- Have you ever heard of “leap days” ?
- How about “leap seconds” !?

Dates

- Four options:
  - LocalDate, YearMonth, MonthDay, and Year

- LocalDate date = LocalDate.of(2000, Month.NOVEMBER, 20);
  // date object for November 20, 2000

- DayOfWeek day = date.getDayOfWeek();

- YearMonth date2 = YearMonth.now();
  // 2013-06

- Similarly for MonthDay
  and for Year

Dates *and* Time

- LocalTime and LocalDateTime
  - “local” because ignores timezones

- LocalTime thisSec = LocalTime.now();
  System.out.print(thisSec.getHour(), thisSec.getMinute(), thisSec.getSecond());

- LocalDateTime myTime = LocalDateTime.of(1994, Month.APRIL, 15, 11, 30);
  System.out.print(myTime);
  - 1994-04-15T11:30:00.985

  - The letter ‘T’ means “time”.
    If you want to avoid it, then you have to print each part sepearately, like above.

Timing

- Count how long something takes
- No import needed!
-
```
double start = System.currentTimeMillis();
// … do some stuff
double end = System.currentTimeMillis();
double totalSeconds = (end – start) / 1000;
```