Lecture 0.5
INTRPRG Review

Because things are only going to get crazier later

Methods

A way to reuse your code!

Declaring (and using) Methods

Create a method named

`getDaysInMonth`

that accepts an

int parameter representing a month

(January=1 … December=12).

This method

returns the number of days

in the given month.

Assume that February always has 28 days (ignore leap years).

What the method **needs** to get the job **done**

getDaysInMonth int parameter month returns number of days

```
public static int getDaysInMonth(int month){
        // your algorithm code here
}
```

Solving the problem

```
public static int getDaysInMonth(int month){
        // to store the result
        int days = 0;

        // solve the problem using the given parameter(s)
        switch(month){
                …
        }

        // send back the result
        return days;
}
```

Assume this contains a value (it is a "given")

The expression after **return** must match the return **type**

Methods are made to be **reusable**

```
public static int getDaysInMonth(int month){
        …
}

public static void getTotalDays(int start, int end){
        …
        total += getDaysInMonth(i);
        …
}

public static void main(){
        …
        while(days > getDaysInMonth(temp))
        …
}
```

You "call" the method to run its code.

You pass an argument, whose value is assigned to the parameter

The value of the result gets returned by (or as) the method call

We can use the method again, this time for a different problem!

Pro Tip: The more you **reuse** a method, the more **"sulit"** it is.

Programming Algorithms

Transforming your ideas into code

How to come up with an **algorithm**?

1) Isolate the **goal**

Create a method named `getDayOfMonth`... This method returns the **day of the month** after the given number of days has passed.

2) Take an inventory of your **givens**

- Start month and day (both int)
- Number of days after (int)
- Knowledge of number of days per month (getDaysInMonth)

Start by doing it **manually**

So, we start from a certain month-day...

1/22 (January 22)

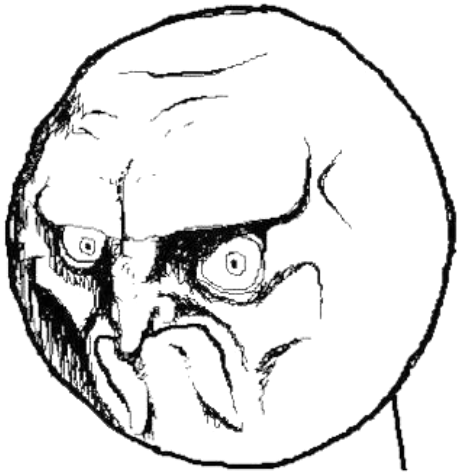
And need to move past a certain number of days...

1/22 + 3

Hmmm, let's try that...

1/22 + 3 = 1/25

The answer is **25**.



Done! Easiest problem ever! **NO.**

Always consider or try out other cases before finalizing your solution.
It might not work for all valid givens.

Start by doing it **manually** (again)

So, we start from a certain month-day…

1/22 (January 22)

And need to move past a certain number of days…

1/22 + 51

Hmmm, let's try that…

1/22 + 51 = 1/73

Wait, that's not right…

January only has 31 days and 1/73 > 31

If we go over 31 days in January, we move to February…

1/32 → 2/1 , where x/32 – x/31 → 1 and 1/x → 2/x

Continue finding the **solution**

Okay, let's try that…

1/73 – 31 → 2/42

Aww man, closer but still not right… just try and try

2/42 – 31 → 3/10

Wait! 31? What was 31 again?

January = 31 days, but February = 28!

Have to remember what those values mean…

The right way should be…

2/42 – 28 → 3/14

That looks about right! Good!

If it's not okay, then we can just repeat what we did before.

Create a step-by-step plan or algorithm

- Start with month/day is 1/22, need to go 51 days after
- 1/22 + 51 = 1/73
- 1/73 is over the max number of days of month 1 (31)
- 1/73 – 31 = 2/42
- 2/42 is still over the max number of days for month 2 (28), so we repeat
- 2/42 – 28 = 3/14
- 3/14 is within the limit of month 3 (31)
- Otherwise, we just repeat until it's within limit

We've reached our desired solution (and result)!

Convert your algorithm into Java code

- Start with **month/day** is 1/22, need to go 51 days **after**

```
getDayOfMonth(int month, int day, int after)
```

- 1/22 + 51 = 1/73

```
day += after;
```

- 1/73 is over the max number of days of month 1 (31)

```
if(day > getDaysInMonth(month))
```

- 1/73 – 31 = 2/42

```
day -= getDaysInMonth(month)
month++;
```

- 2/42 is still over the max number of days for month 2 (28) so we repeat

```
while(day > getDaysInMonth(month))
```

- 2/42 – 28 = 3/14

```
day -= getDaysInMonth(month)
month++;
```

We've reached our desired result!

```
return day;
```

Wait! We still need to clean up our code

```
public static int getDayOfMonth(int month, int day, int after){
        day += after;
        if(day > getDaysInMonth(month)){
                day -= getDaysInMonth(month);
                month++;
                while(day > getDaysInMonth(month)){
                        day -= getDaysInMonth(month);
                        month++;
                }
        }
        return day;
}
```

This is redundant.
Why?
The **while** statement is already like a repeating **if** statement!
We don't really need the outer **if statement**.

Fixed and Simplified!

```
public static int getDayOfMonth(int month, int day, int after){
        day += after;
        while(day > getDaysInMonth(month)){
                day -= getDaysInMonth(month);
                month++;
                // find what else can go wrong here?
        }
        return day;
}
```

We didn't use any locally declared variables here, but you can use some if that makes things easier.

Just remember that your parameters are also local variables. They just contain important "given" data from the start.

So, don't overwrite them without first making good use of them.

Let's move on to some more challenging problems

- Ask the user for 5 ints, and then display the sum of the numbers.

Oh wait, it was supposed to be more challenging

- Ask the user for 5 ints, and then display the sum of the numbers. Then, display these numbers from highest to lowest.

And even **more** challenging.

- Ask the user for a number ***n***. Ask the user for ***n*** ints. Display the sum of the numbers. Then, display these numbers from highest to lowest.

How many variables will it take to store **n** values?
We can't just create more variables <u>while the program is running</u>.
So, this is impossible with the basic tools you learned in INTRPRG.

Arrays

We're dealing with lots of stuff now.

Let's compare

Variable →

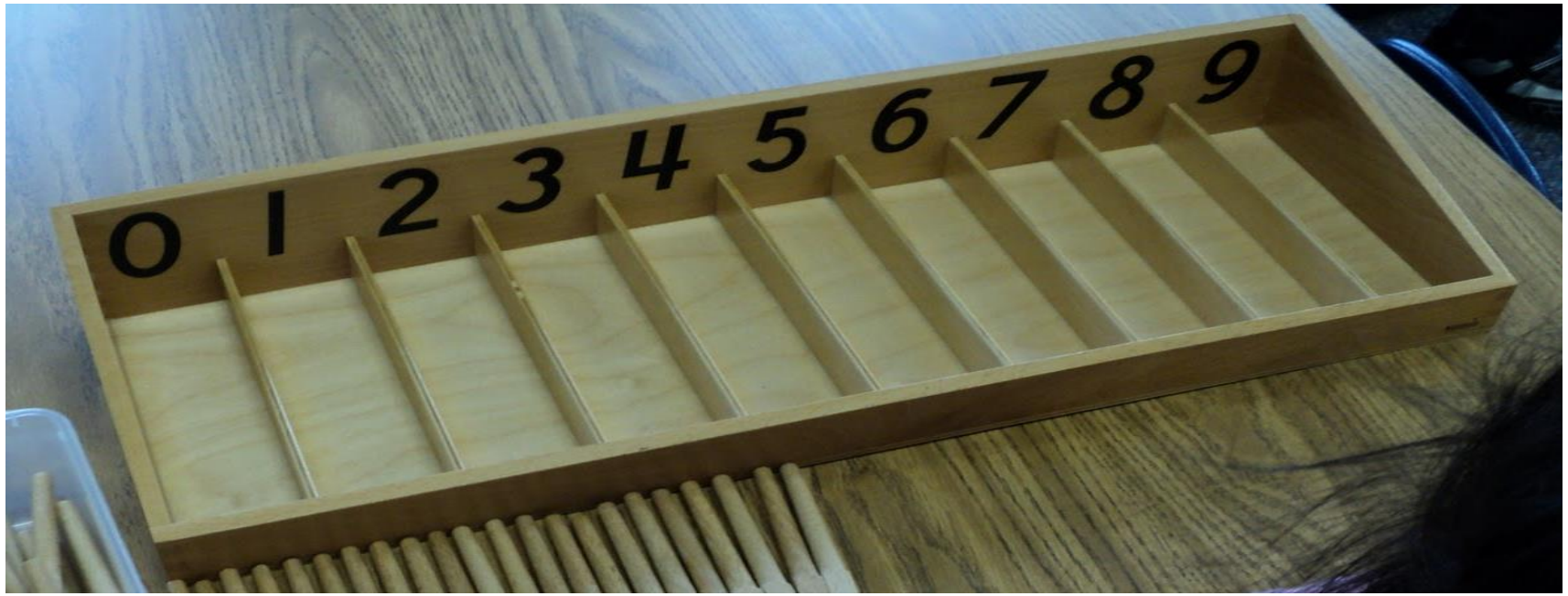
Array→

Do some **Research**

- Use Google or the Oracle website for resources on how to **declare** and **access** arrays.
- Keep those tabs open before we proceed to the next few slides.

Define, what is an Array?

A **container** that holds a **fixed number** of values of a **single type**



Learning some Java Array **Syntax**

Declaring

```
<data type>[] <identifier>
```

Declaring and Initializing

```
<data type>[] <identifier> = new
<data type>[<size>];
```

Converting words and ideas into Arrays

- I need 10 ID Numbers

```
int[] AIDNumbers = new int[10];
```

- I need 100 Student Names

```
String[] AStudentNames = new String[100];
```

Assigning values to each element

- Index of elements starts from 0 up to (array length– 1)

```
int[] Array_Integers = new int[10];

Array_Integers[0] = 1990;
Array_Integers[1] = 1991;
Array_Integers[2] = 1992;
Array_Integers[3] = 1993;
Array_Integers[4] = 1994;
Array_Integers[5] = 1995;
Array_Integers[6] = 1996;
Array_Integers[7] = 1997;
Array_Integers[8] = 1998;
Array_Integers[9] = 1999;
```

Accessing array elements

- Use the element's index to get a specific int from an Array of ints, or a String from an Array of Strings.

```
System.out.println (Array_Integers[0]);
System.out.println (Array_Integers[1]);
System.out.println (Array_Integers[2]);
System.out.println (Array_Integers[3]);
System.out.println (Array_Integers[4]);
System.out.println (Array_Integers[5]);
System.out.println (Array_Integers[6]);
System.out.println (Array_Integers[7]);
System.out.println (Array_Integers[8]);
System.out.println (Array_Integers[9]);
System.out.println (Array_Integers[10]);
```

Combining operations

- Doing integer operations with ints in an Array of ints.

```
Array_Integers[0] = Array_Integers[1] + 2;
Array_Integers[5] = Array_Integers[6] +
Array_Integers[7];
```

Getting the array length

- What is the size of any given Array?
- Dunno, ask the Array.

```
System.out.print(Array_Integers.length);
//Array_Integers says 10
```

Accessing **all the elements** at once

- Arrays have a lot of stuff, so we use loops
- Printing all of the elements in the array:

```
int i;
for(i = 0; i < Array_Integers.length; i++)
    System.out.println(Array_Integers[i]);
```

What about other **set operations**?

```
for(int i = 0; i < AIDNumbers.length; i++){
        ??? AIDNumbers[i] ???
}
```

Can you think of other stuff to do with this array?

Try the challenge problem again.

- Ask the user for 5 ints, and then display the sum of the numbers. Then, display these numbers from highest to lowest.

- Ask the user for a number ***n***. Ask the user for ***n*** ints. Display the sum of the numbers. Then, display these numbers from highest to lowest.