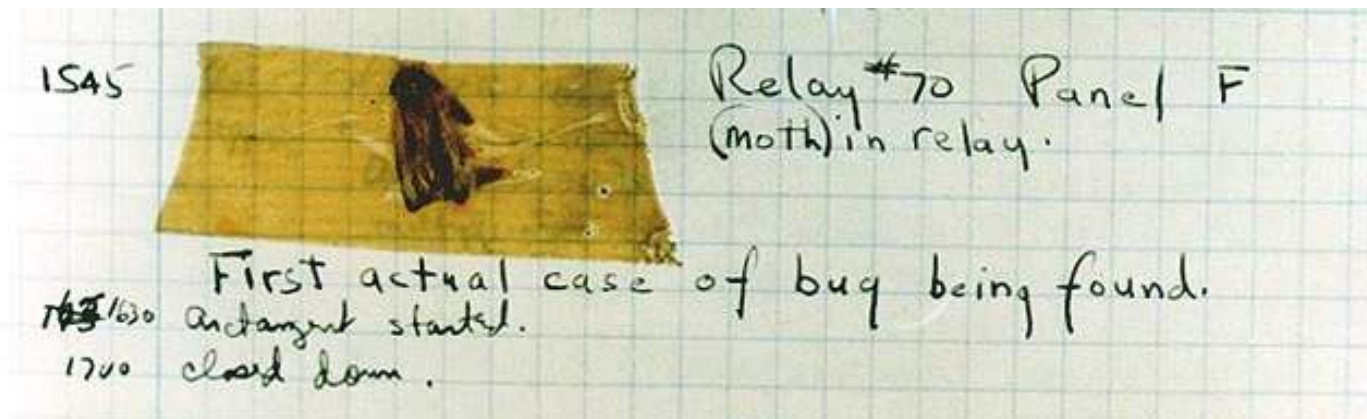


# More on variables, arrays, debugging



# Overview

- Variables revisited
  - Scoping
- Arrays revisited
- Debugging
  - Tip and tricks to help you keep your sanity

# Variable scoping

- Variables live within their curly braces
  - Once curly brace block finishes, variable is gone!

```
public class DoStuff
{
    public static void main(String [] args)
    {
        int x = 0;
        for (int y = 0; y < 5; y++)
        {
            x = x + y;
        }
        x = x * y;
    }
}
```

y only lives in the for-loop

y is undefined, this won't compile!

# Variable scoping

- You can declare same name again
  - But only after no longer “in scope”

```
public class DoStuff
{
    public static void main(String [] args)
    {
        int x = 0;
        for (int y = 0; y < 5; y++)
        {
            x = x + y;
        }
        int y = 1;
        x = x * y;
    }
}
```

# Arrays revisited

- Arrays

- Store a bunch of **values under one name**
- **Declare and create in one line:**

```
int N = 8;  
int [] x = new int[10];  
double [] speed = new double[100];  
String [] names = new String[N];
```

- To get at values, use name and index between []:

```
int sumFirst2 = x[0] + x[1];  
speed[99] = speed[98] * 1.1;  
System.out.println(names[0]);
```

- **Array indexes start at 0!**

# Arrays revisited

- Arrays

- You can just declare an array:

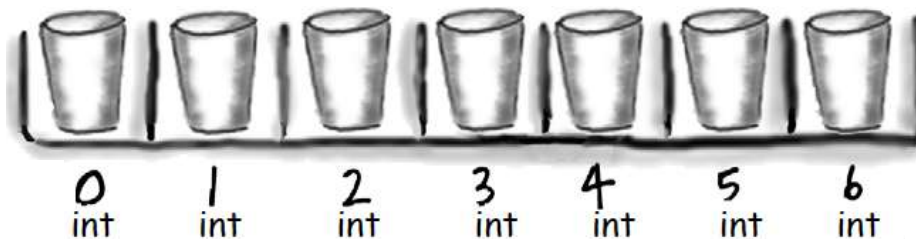
```
int [] x;
```

- But `x` is not very useful until you “new” it:

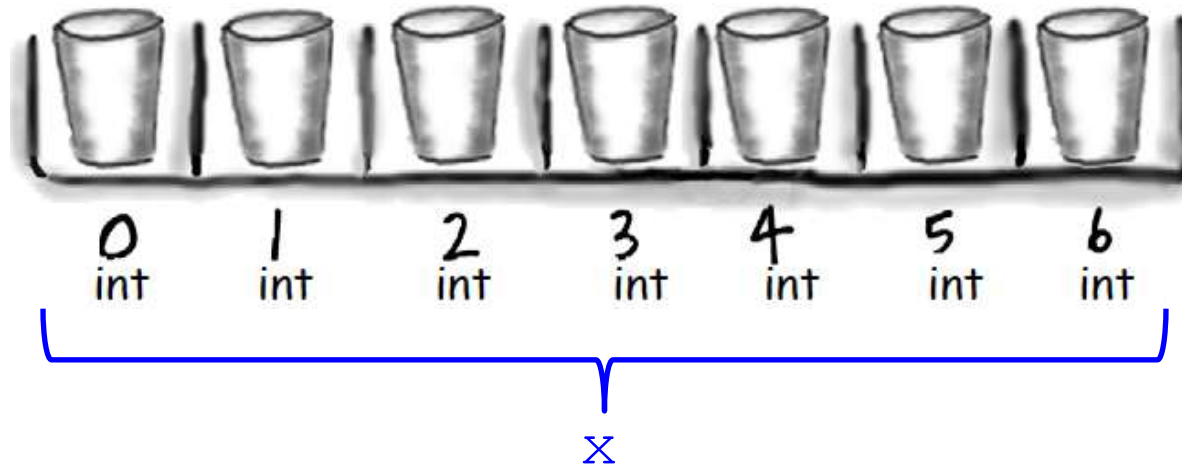
```
int [] x;  
x = new int[7];
```

- `new` creates the memory for the slots

- Each slot holds an independent `int` value
- Each slot initialized to default value for type

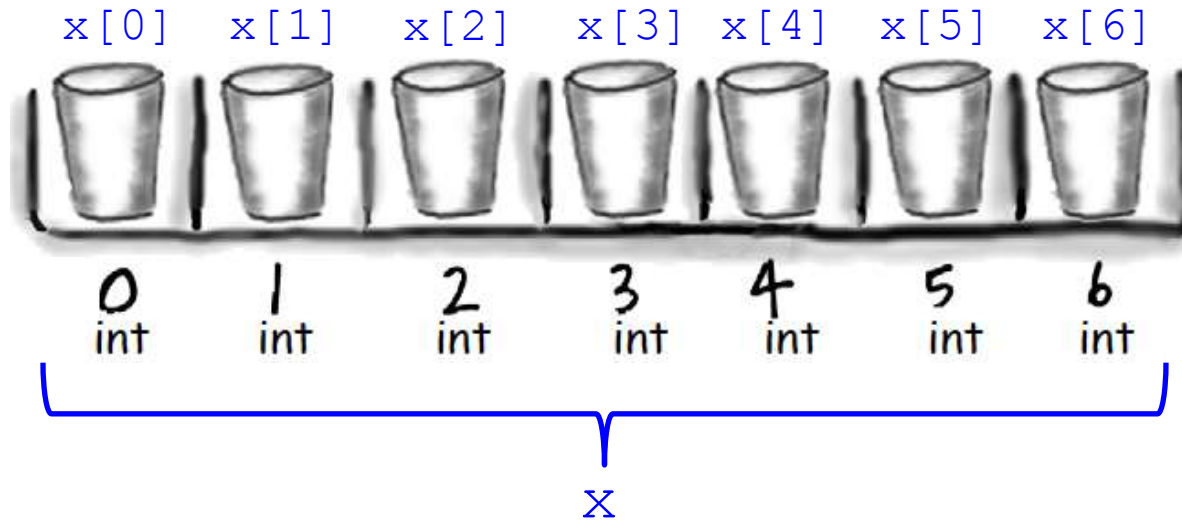


# Arrays revisited



- **x** refers to the whole set of slots
- You can't use the variable **x** by itself for much
- Except for finding out the number of slots: **x.length**

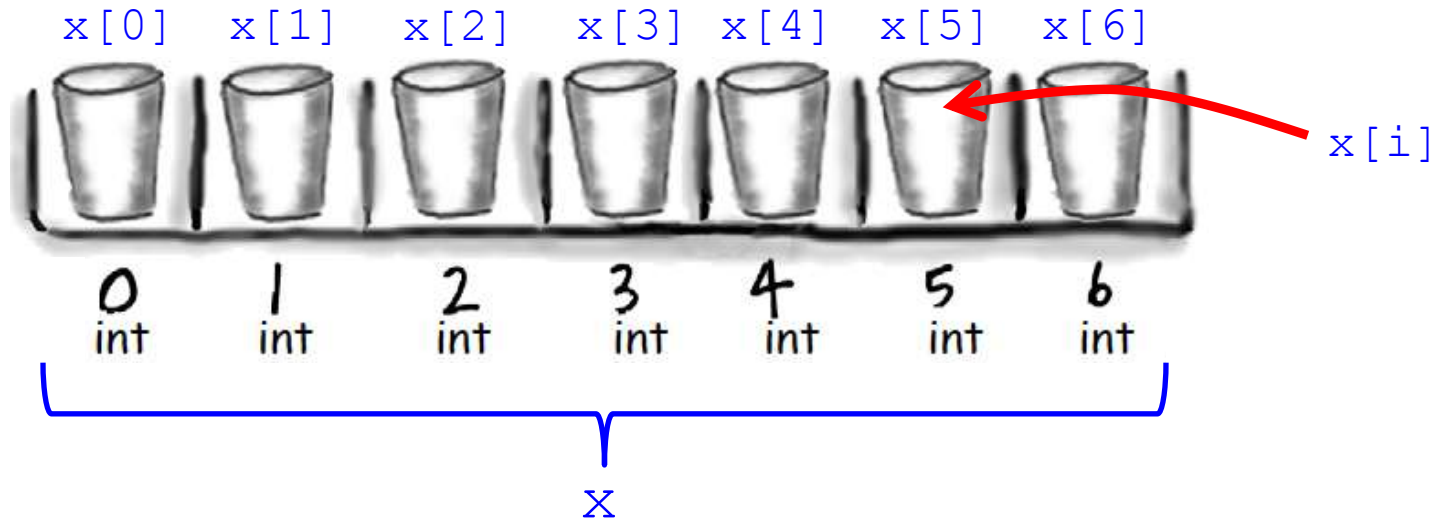
# Arrays revisited



- `x[0]`, `x[1]`, ..., `x[6]` refers to the value at a particular slot
- `x[-1]` or `x[7]` is an `ArrayIndexOutOfBoundsException`



# Arrays revisited



- $x[i]$  refers to the value at a slot, but the **slot index is determined by variable  $i$** 
  - if  $i = 0$  then  $x[0]$ , if  $i = 1$  then  $x[1]$ , etc.
- Whatever is **inside [] must be an int**
- Whatever is **inside [] must be from 0 to  $x.length - 1$  (inclusive)**

# Debugging

- Majority of program development time:
  - Finding and fixing mistakes! a.k.a. **bugs**
  - It's not just you: **bugs happen to all programmers**


9/9

0800 Antan started  
 1000 " stopped - antan ✓  
 13'02 (032) MP-MC { 1.2700 9.037 847 025  
 032) PRO 2 2.130476415 (2) 9.037 846 995 correct  
 correct 2.130676415

Relays 6-2 in 032 failed speed test  
 in relay " 11.00 test.

Relays changed

1100 Started Cosine Tape (Sine check)  
 1525 Started Multi Adder Test.

1545  Relay #70 Panel F  
 (Moth) in relay.

First actual case of bug being found.  
 1650 Antan started.  
 1700 closed down.

failing  
2145  
11.00 test

# Debugging

- Computers can help find bugs
  - But: computer can't automatically find all bugs!
- Computers do **exactly what you ask not necessarily what you want**
- There is always a **logical explanation!**
  - Make sure you **saved & compiled** last change



“As soon as we started programming, we found out to our surprise that it wasn't as easy to get programs right as we had thought. Debugging had to be discovered. I can remember the exact instant when I realized that a large part of my life from then on was going to be spent in finding mistakes in my own programs.”

*-Maurice Wilkes*



“There has never been an unexpectedly short debugging period in the history of computers.”

*-Steven Levy*

# Preventing bugs

- Have a plan
  - Write out steps in English before you code
  - Write comments first before tricky bits
- Use good coding style
  - Good variable names
    - If variable is called `area` it should hold an area!
  - Split complicated stuff into manageable steps
  - ()'s are free, force order of operations you want
  - Carefully consider loop bounds
- Listen to Eclipse (IDE) feedback

# Finding bugs

- How to find bugs
  - Add **debug print statements**
    - Print out state of variables, loop values, etc.
    - Remove before submitting
  - **Use debugger** in your IDE
    - Won't work for programs using file redirection
  - **Talk through program** line-by-line
    - Explain your program to a novice



# Debugging example

- Problem:

- Given an integer  $N > 1$ , compute its prime factorization

- $98 = 2 \times 7 \times 7$

- $17 = 17$

- $154 = 2 \times 7 \times 11$

- $16562 = 2 \times 7 \times 7 \times 13 \times 13$

- Possible application: **Break RSA encryption** (used in Internet commerce)

# A simple algorithm

- **Problem:**

- Given an integer  $N > 1$ , compute its prime factorization

- **Algorithm:**

- Starting with  $i=2$ , repeatedly divide  $N$  by  $i$  as long as it evenly divides, output  $i$  every time it divides
- Increment  $i$
- Repeat

# Example run

i	N	Output
2	16562	2
3	8281	
4	8281	
5	8281	
6	8281	
7	8281	7 7
8	169	
9	169	
10	169	
11	169	
12	169	
13	169	13 13
14	1	
...	1	



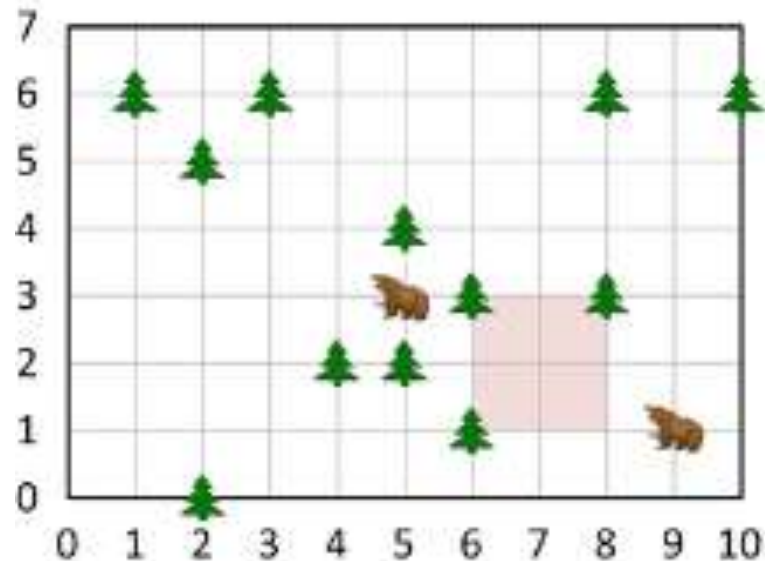
# Buggy factorization program

```
public class Factor
{
    public static void main(String [] args)
    {
        long n = Long.parseLong(args[0])
        for (i = 0; i < n; i++)
        {
            while (n % i == 0)
                System.out.print(i + " ")
                n = n / i
        }
    }
}
```

***This program has many bugs!***

# Incremental development

- Split development into stages:
  - Test thoroughly after each stage
    - Don't move on until it's working!
  - Example: LoggingLease



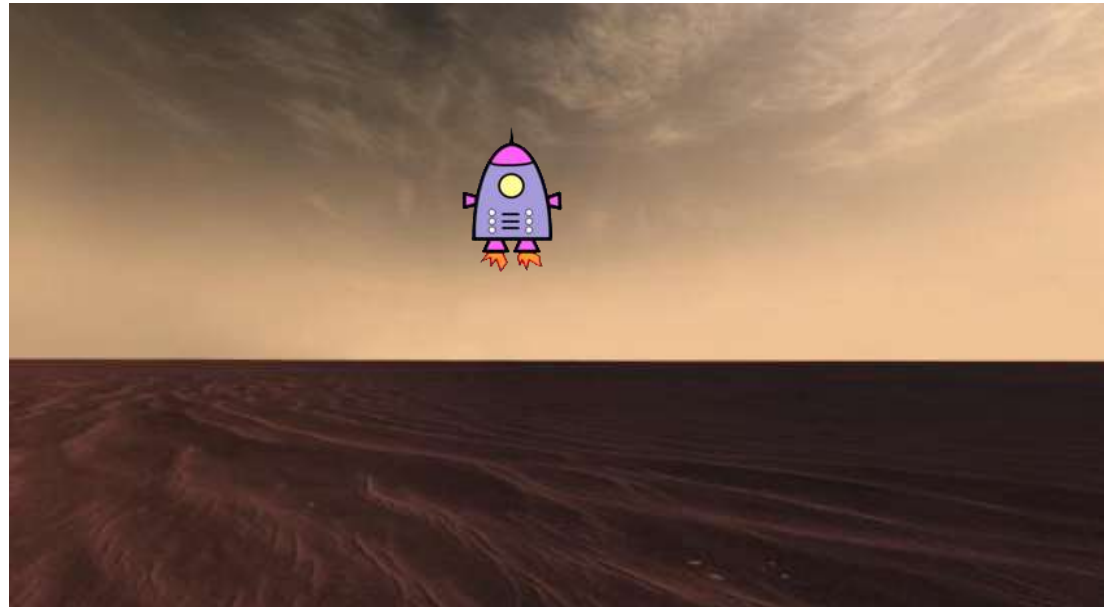
# LoggingLease development

- One possible set of **development stages**:

Stage	Goal	Testing
1	Parse command line option, calculate how big lease is on each side.	Print out side length (temporary).
2	Read height, width and number of points from StdIn.	Print height, width, and number points (temporary).
3	Declare, create and read data into arrays from StdIn. Count trees and bear dens.	Print out number of trees and bear dens.
4	Loop over all valid southwest corners for leases.	Print out each SW corner (temporary).
5	For each valid lease, find all points in that lease.	For each SW corner, print all points in that lease (temporary).
6	Add logic to find total cubic feet of wood in a given lease and to determine if a bear den is in the lease.	For each SW corner, print out wood total and if there is a bear.
7	Add logic to keep track of best bear-free lease. Output final result.	Compare with our sample outputs.

# Incremental development

- Split development into stages
  - Test thoroughly after each stage
  - Example: MarsLander  
<http://www.youtube.com/watch?v=Ene3LYsRMco>



# MarsLander development

- One possible set of **development stages**:

Stage	Goal	Testing
1	Read in data from file into variables	Print things back out to console
2	Load the images, setup drawing window, draw background, ship, landing pad.	Static lander in correct position on top of Mars background
3	Add physics so lander starts moving	Lander attracted to ground and just keeps on going
4	Add in keyboard input, adjust velocity based on input	Lander that can be flown around
5	Change lander image based on keyboard input, play thruster sound, update fuel counter	Lander can be flown around with visual and audio feedback, thrusters disabled once out of fuel
6	Detect contact with ground, change image and play sound for good or bad landing	Complete game.

# Summary

- Variables
  - Live within their curly braces
- Arrays
  - Hold a set of independent values of same type
  - Access single value via index between []'s
- Debugging
  - Have a plan before coding, use good style
  - Learn to trace execution
    - On paper or with print statements
  - Incremental development