Welcome Back from Spring Break

Today's Material

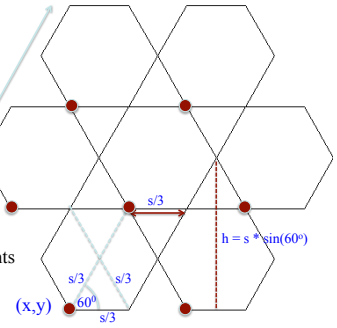
- All of Chapter 7
  - Continuing loops discussion
  - Will conclude Thursday
- **Today's Lab**: For Loops
  - Requires that you remember the syntax from before break
  - Also uses some of today's material for problem solving
- Class is getting easier…

Assignments

- Assignment A4 now graded
  - **Completion Time:**
    - Mean 6.7 hrs; Median 6 hrs
    - Max: 30 hrs; Min: 1 hr
  - **Grades:**
    - Mean 95.1, Median 100
- Assignment A5 posted
  - Due week from Thurs
  - Note the choice of problems

Grisley Snowflakes

- Given (as shown):
  - Length s
  - Point (x,y)
- Find:
  - Coordinates of all red points
- Draw:
  - Snowflakes of one less depth and size s/3 at those points



Today's Terminology

- **assertion**: true-false statement placed in a program to *assert* that it is true at that point
  - Can either be a comment, or a special Java command
- **precondition**: assertion placed before a statement
  - Same idea as **method precondition**, but more general
- **postcondition**: assertion placed after a statement
- **loop invariant**: assertion supposed to be true before and after each iteration of the loop
  - Distinct from **class (field) invariant**
- **iteration of a loop**: one execution of its repetend

Review: Assert Statements

```
assert <boolean>;  // Creates Exception if <boolean> false
assert <boolean> : <String>; // As above, but displays <String>
```

- Can write and forget
  - Only used if debugging turned on in Java
  - Otherwise, Java treats it like a comment
- Code defensively!

```
/** Set worker's last name to n
 * Precondition: n cannot be null */
public void setName(String n) {
  assert n != null;
  lname = n;
}
```

Comment form of the assertion

Language support for an assertion

Assertions versus Asserts

- Assertions **prevent bugs**
  - Help you keep track of what you are doing
- Also **track down bugs**
  - Make it easier to check belief/code mismatches
- Do not confuse w/ asserts
  - All asserts are assertions
  - But reverse is not true
  - Cannot always convert a comment to an assert

```
// x is the sum of 1..n
```

Comment form of the assertion.

x	?	n	1
x	?	n	3
x	?	n	0

Preconditions & Postconditions

precondition

```
// x  = sum of 1..n-1
x = x + n;
n = n + 1;
// x =  sum of 1..n-1
```

postcondition

n
1 2 3 4 5 6 7 8

x contains the sum of these (6)

n
1 2 3 4 5 6 7 8

x contains the sum of these (10)

- **Precondition:** assertion placed before a segment
- **Postcondition:** assertion placed after a segment

Meaning

If precondition is true, then postcondition will be true

Invariants: Assertions That Do Not Change

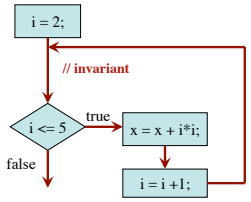
- **Loop Invariant**: an assertion that is true before and after each iteration (execution of repetend)

```
x = 0;
for (int i = 2; i <= 5; i= i +1) {
      x = x + i*i;
}
// x = sum of squares of 2..5
```

Invariant:

x = sum of squares of 2..i-1

in terms of the range of integers that have been processed so far



The loop processes the range 2..5

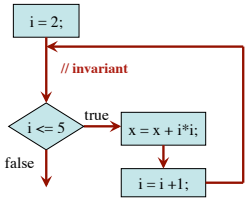
Invariants: Assertions That Do Not Change

```
x = 0;
// Inv: x = sum of squares of 2..i-1
for (int i = 2; i <= 5; i= i +1) {
    x = x + i*i;
}
// Post: x = sum of squares of 2..5
```

Integers that have been processed:	2, 3, 4, 5
Range 2..i-1:	2..5

Invariant was always true just before test of loop condition. So it's true when loop terminates

x	~~0~~ ~~4~~ ~~9~~ ~~16~~ 54
i	~~2~~ ~~3~~ ~~4~~ ~~5~~ 6



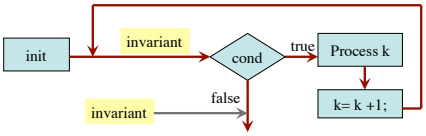
The loop processes the range 2..5

Designing For-Loops

Command to do something

```
// Process integers in a..b
// inv: the integers in a..k-1 have been processed
for (int k = a;  k <= b;  k = k + 1) {
    Process integer k;
}
// post: the integers in a..b have been processed
```

Equivalent postcondition



Methodology for Making a For-Loop

1. Recognize that a range of integers b..c has to be processed
2. Write the command and equivalent postcondition
3. Write the basic part of the for-loop
4. Write loop invariant
5. Figure out any initialization
6. Implement the repetend (Process k)

```
// Process b..c
Initialize variables (if necessary) to make invariant true
// Invariant: range b..k-1 has been processed
for (int k= b;  k <= c;  k= k+1) {
    // Process k
}
// Postcondition: range b..c has b
```

Finding an Invariant

Command to do something

```
// Store in b the value of : "no int in 2..n-1 divides n"
b = true;
// invariant: b = no int in 2..k-1 divides n
for (int k = 2; k < n;  k = k +1) {
    // Process k;
    if (n%k == 0) b = false;
}
// b = "no int in 2..n-1 divides n"
```

Equivalent postcondition

What is the invariant? 1 2 3 … k-1 k k+1 … n