Reminders

Lab 0.1 has been released

- Read the assignment, the reading materials and set-up Eclipse & SVN on your computer (from Stellar).
- Labs are scheduled from 1pm-4pm in 32-123.
- Bring your laptop for the lab. If you don’t own one, contact the staff.

Problem Set 1

- Will be released on Monday (Sept 13), and will be due on the next Wednesday (Sept 22).
- Start early, and remember to read the collaboration policy carefully.

New to Java?

- See the “Materials” section of the class website for useful references.
Today

Recap: Class Logistics & Lecture 1
- Advice for doing well in 6.005
- Hailstone Sequences, Basic Java

Basic Eclipse Use & Debugging
- Extend Hailstone sequences.
- Debugging using Breakpoints, Conditional Breakpoints and Runtime Assertions.
The essence of 6.005

Before 6.005, you were able to write small-scale yet useful programs.
The essence of 6.005

Without 6.005, you may end up developing solutions that “work” on larger scales, but...

...
The essence of 6.005

Without 6.005, you may end up developing solutions that “work” on larger scales, but clearly leave a lot to be desired.
Advice for 6.005

Software Quality

➤ 6.005 is the only class in MIT in which the code you write will be critiqued for its quality, and not just its correctness.

Solve problems Iteratively

➤ Think before you code: it is not sufficient to simply pass test cases, or meet the specification you are provided.
➤ Ask the staff for feedback on your designs!

Learn Real-World Software Practices

➤ Write self-commenting code.
➤ Use defensive programming and decouple components.
➤ Don’t reinvent the wheel: use Java libraries.
Recap: Lecture 1

**Hailstone Sequences**

- Starting at $n$, the next number in the sequence is:
  - $n/2$ if $n$ is even,
  - $3n + 1$ if $n$ is odd.
- Sequence oscillates between high and low numbers until $n$ becomes 1.
public class Hailstone {
    public static void main(String[] args) {
        ...
    }
    public static String hailstoneSequence(int n) {
        ...
    }
    public static int hailstoneLength(int n) {
        ...
    }
    public static int[] hailstoneArray(int n) {
        ...
    }
}
Java Program for Hailstones

```java
public class Hailstone {
    public static void main(String[] args) {
        ...
    }
    public static String hailstoneSequence(int n) {
        ...
    }
    public static int hailstoneLength(int n) {
        ...
    }
    public static int[] hailstoneArray(int n) {
        ...
    }
}
```

Java code must be enclosed in a class. The class name must be the same as the file name. So this class would be in `Hailstone.java`
public class Hailstone {

    public static void main(String[] args) {
        ... 
    }

    public static String hailstoneSequence(int n) {
        ... 
    }

    public static int hailstoneLength(int n) {
        ... 
    }

    public static int[] hailstoneArray(int n) {
        ... 
    }
}

The entry point for Java programs... for now, just memorize what it looks like.
Java Program for Hailstones

```java
public class Hailstone {
    public static void main(String[] args) {
        ...
    }

    public static String hailstoneSequence(int n) {
        ...
    }

    public static int hailstoneLength(int n) {
        ...
    }

    public static int[] hailstoneArray(int n) {
        ...
    }
}

These static methods can be called inside the main function. Breaking code into small methods improves readability.
```
/**
 * Returns a comma-separated String that contains the
 * Hailstone sequence starting at n.
 * e.g. n=3 -> "3,10,5,16,8,4,2,1"
 */

public static String hailstoneSequence(int n) {
    String seq = String.valueOf(n);
    while (n != 1) {
        if (n % 2 == 0) {
            n = n / 2;
        } else {
            n = 3 * n + 1;
        }
        seq = seq + ""," + n;
    }
    return seq;
}
/**
 * Returns a comma-separated String that contains the
 * Hailstone sequence starting at n.
 * e.g. n=3 -> "3,10,5,16,8,4,2,1"
 */

public static String hailstoneSequence(int n) {
    String seq = String.valueOf(n);
    while (n != 1) {
        if (n % 2 == 0) {
            n = n / 2;
        } else {
            n = 3 * n + 1;
        }
        seq = seq + ""," + n;
    }
    return seq;
}
/**
 * Returns a comma-separated String that contains the Hailstone sequence starting at n.
 * e.g. n=3 -> "3,10,5,16,8,4,2,1"
 */

public static String hailstoneSequence(int n) {
    String seq = String.valueOf(n);
    while (n != 1) {
        if (n % 2 == 0) {
            n = n / 2;
        } else {
            n = 3 * n + 1;
        }
        seq = seq + "," + n;
    }
    return seq;
}
Click link.

Scroll down to methods.
Just hovering over the statement can also yield the same information.
//**
* Returns a comma-separated String that contains the
* Hailstone sequence starting at n.
*/

public static String hailstoneSequence(int n) {
    String seq = String.valueOf(n);
    while (n != 1) {
        if (n % 2 == 0) {
            n = n / 2;
        } else {
            n = 3 * n + 1;
        }
        seq = seq + "", " + n;
    }
    return seq;
}
/* *
 * Returns a comma-separated String that contains the
 * Hailstone sequence starting at n.
 */

public static String hailstoneSequence(int n) {
    String seq = String.valueOf(n);
    while (n != 1) {
        if (n % 2 == 0) {
            n = n / 2;
        } else {
            n = 3 * n + 1;
        }
        seq = seq +"," + n;
    }
    return seq;
}
Java Program for Hailstones

/**
 * Returns a comma-separated String that contains the
 * Hailstone sequence starting at n.
 */

public static String hailstoneSequence(int n) {
    if (n == 1)
        return "1";
    else if (n % 2 == 0)
        return n + "," + hailstoneSequence(n/2);
    else
        return n + "," + hailstoneSequence(3*n+1);
}
/**
 * Returns the length of the Hailstone sequence starting
 * at n (including 1).
 */

public static int hailstoneLength(int n) {
    int lengthSoFar = 1;
    while (n != 1) {
        if (n % 2 == 0) {
            n = n / 2;
        } else {
            n = 3 * n + 1;
        }
        lengthSoFar++;
    }
    return lengthSoFar;
}
Java Program for Hailstones

/**
 * Returns the length of the Hailstone sequence starting
 * at n (including 1).
 */

public static int hailstoneLength(int n) {
    if (n == 1)
        return 1;
    else if (n % 2 == 0)
        return 1 + hailstoneLength(n/2);
    else
        return 1 + hailstoneLength(3*n+1);
}

Solution
/**
* Returns the maximum element in an array of positive integers.
*/

public static int maxValue(int[] array) {
    int max = 0; // smallest negative integer

    for (int v : array) {
        if (v > max) max = v;
    }

    return max;
}
Java Program for Hailstones

/**
 * Returns the maximum element in an array of positive integers.
 */

public static int maxValue(int[] array) {
    int max = 0; // smallest negative integer
    for (int v : array) {
        if (v > max) max = v;
    }
    return max;
}

Note that method names should generally start with a verb, such as:
computePrincipal(...) or reduceBalance(...). Classes should be named
starting with a capital letter (e.g. Person, Car). Methods, local variables and
arguments must start with the first letter in lower case (e.g. maxIndex,
totalProfit).

So this method can be made more readable.
Lists in Java

/**
 * Returns an array of numbers representing the sequence starting at n
 */

public static int[] hailstoneArray(int n) {
    int[] sequence = new int[hailstoneLength(n)];
    sequence[0] = n;

    int i = 1;
    while (n != 1) {
        if (n % 2 == 0) {
            n = n / 2;
        } else {
            n = 3*n+1;
        }
        sequence[i] = n; i++;
    }
    return sequence;
}
Lists in Java

/**
 * Returns a list of numbers representing the sequence starting at n.
 */

public static ArrayList<Integer> hailstoneList(int n) {
    ArrayList<Integer> sequence = new ArrayList<Integer>();
    sequence.add(n);
    while (n != 1) {
        if (n % 2 == 0) {
            n = n / 2;
        } else {
            n = 3*n + 1;
        }
        sequence.add(n);
    }
    return sequence;
}
Hash Maps in Java

Hailstone Sequences can Overlap

- Hailstone 99:
  99, 298, 149, 448, 224, 112, 56, 28, 14, 7, 22, 11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1

- Hailstone 100:
  100, 50, 25, 76, 38, 19, 58, 29, 88, 44, 22, 11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1

Memoization: avoid re-computing results

- After we compute hailstone (99), we should be able to reuse some of our previous work for hailstone (100).
- Main Idea:
  - When you wish to compute hailstone(n), check if the result is already in a Table. If it is, just use the stored result.
  - If the result is not in the Table, compute result and update the Table before returning.
  - In Python, we would use a dict. In Java, we use a HashMap.
  - HashMap<key_type, value_type> maps objects of type key_type to those of value_type, just like a dict.
public class Hailstone {
    public static HashMap<Integer, String> storedSequence
        = new HashMap<Integer, String>();

    public static String hailstoneSequence(int n) {
        if (storedSequence.containsKey(n)) {
            return storedSequence.get(n);
        }

        String seq;
        if (n == 1)
            seq = "1";
        else if (n % 2 == 0)
            seq = n + "," + hailstoneSequence(n / 2);
        else
            seq = n + "," + hailstoneSequence(3*n + 1);

        storedSequence.put(n, seq);
        return seq;
    }
}
Debugging Java

**Use Print Statements**
- `System.out.println(…)` can be used to tell you what your program is doing.

**Use Eclipse ‘Debug’ Mode**
- Add Breakpoints
- Step Through Code
- Use Conditional Breakpoints

**Use Runtime assertions**
- Assert something, and add “–ea” to VM arguments
Using Breakpoints

Right click on gray bar to left of line you want debugger to break at; select “Toggle Breakpoint”

To run in debug mode, right click class, choose “Debug As” > “Java Application”

Source: 6.005 Spring 2010
Conditional Breakpoints

Right click on breakpoint and select “Breakpoint Properties”

Check “Enable Condition” and enter conditional expression in text box

Source: 6.005 Spring 2010
Enabling Assertions (1)

Right click class, select “Run As” > “Run Configurations”
Enabling Assertions (2)

On “Arguments” tab, add “-ea” to the VM (virtual machine) arguments

You can now add assertions in your code in the form “assert expression;”.

Source: 6.005 Spring 2010