

Introduction to Computational Thinking

Lecture #6 (Offline):
Text Input/Output (Scanner);
Object (briefly);
Boolean Expressions; Nested if/else;

Qiao Xiang, Qingyu Song

<https://sngroup.org.cn/courses/ct-xmuf25/index.shtml>

11/7/2025

This deck of slides are heavily based on cs112 at Yale University and cs101 at UCAS, respectively, by courtesy of Dr. Y. Richard Yang and Dr. Zhiwei Xu.

Outline

- ❑ Admin and recap
- ❑ Method w/ return
- ❑ Summary of method definition and invocation rules
 - m overloaded methods
 - m formal arguments are local variables
 - m primitive types use value semantics
- ❑ Text I/O
 - m Input: basic Scanner input
 - m Output: basic printf and String.format

Foundational Programming Concepts

any program you might want to write

objects

methods and classes

graphics, sound, and image I/O

arrays

conditionals and loops

math

text I/O

primitive data types

assignment statements

(Some Potentially Confusing)

Scanner Details

- ❑ The OS will not send input to Scanner constructed using `System.in` until user hits enter (reason?)
- ❑ `nextInt()`, `nextDouble()`, `next()` are **token** based scanning methods
 - m skip **whitespace** (spaces, tabs, new lines) until find first non-white space, collect input into a **token** until a whitespace, send token to the method to interpret; **the following white space remains**
 - m How many tokens appear on the following line of input?
23 John Smith 42.0 "Hello world" \$2.50 " 19"
- ❑ `nextLine()` collects any input character into a string until the first new line and **discards the new line**

Exercise: nextInt, nextLine

```
int i1 = console.nextInt();  
String s1 = console.nextLine();
```



2018↵

result: i1 = 2018 s1 = ""



2018 a ↵

Lets have a try

result: i1 = 2018 s1 = " a "

First read ScannerTokenDiff.java to guess behaviors, then try.

Exercise: nextInt, next

```
int i1 = console.nextInt();  
String s1 = console.next();
```



2018↵

result: i1 = 2018 program hangs, waiting for non-whitespace



2018 □ □ a □ ↵

Lets have a try

result: i1 = 2018 s1 = "a"

First read ScannerTokenDiff.java to guess behaviors, then try.

Exercise: nextLine or next?

```
System.out.print("A number: ");  
int i1 = console.nextInt();  
System.out.print("File name: ");  
String fileName = console.____();
```

Lets have a try

First read ScannerInputExample2.java to guess behaviors, then try.

(Offline) Practice: Scanner Fun

- Please try out ScannerFun.java

ScannerFun.java

Input from File

- There are two approaches
 - m Create a scanner with src as a file (more shortly)
 - m Redirect: a concept in computer science, which allows the operating system to send a file's content as if it is typed in by keyboard:
(Use command line: Terminal -> to working directory)
`%java Plot < USA.txt`

Plot.java USA.txt

Exercise: Plot a Geo Data File

□ File format:

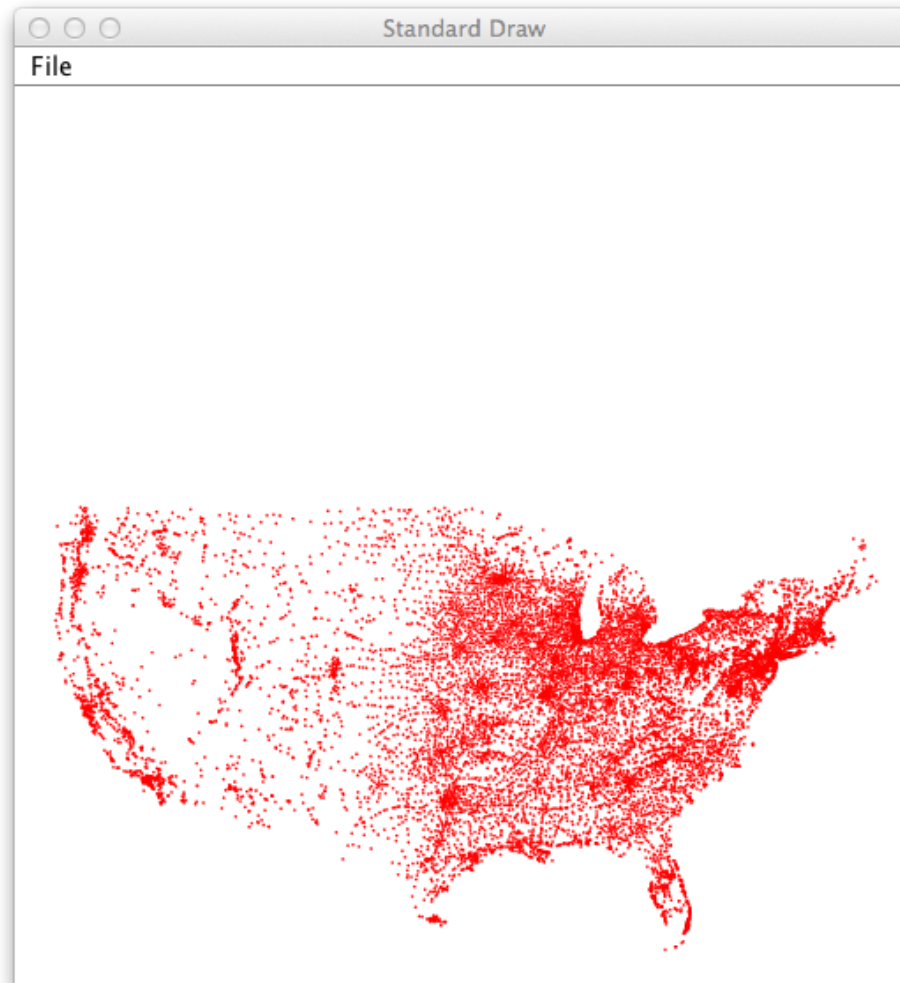
m xmin ymin xmax ymax npoints

m x, y

m ...

Plot.java USA.txt

Input from File



PlotUSA.java USA.txt

Exercise

- ❑ What if you do not want to see the loading process (e.g., see all display at once, not one point at a time)?

Find solution from
StdDraw.java

Plot.java USA.txt

Design Issue

- ❑ What value to return when a token is not the type the scanner expects

```
System.out.print("Which year will you graduate? ");  
Int year = console.nextInt();
```

Output:

Which year will you graduate? **Timmy**



Token and Exception

- When a token is not the type that the scanner expects, since no reasonable (non-ambiguous) return value, Scanner **throws** an **exception** (panic)

```
System.out.print("Which year will you graduate? ");  
int year = console.nextInt();
```

Output:

```
Which year will you graduate? Timmy
```



```
java.util.InputMismatchException
```

```
at java.util.Scanner.next(Unknown Source)  
at java.util.Scanner.nextInt(Unknown Source)  
...
```



Issue: How to avoid crash when user may give wrong type of input?

```
System.out.print("Which year will you graduate? ");  
int year = console.nextInt();
```

Output:

Which year will you graduate? **Timmy**

Approach 1: Test before Proceed

```
System.out.print("Which year will you graduate? ");  
int year = console.nextInt();
```

Output:

Which year will you graduate? Timmy

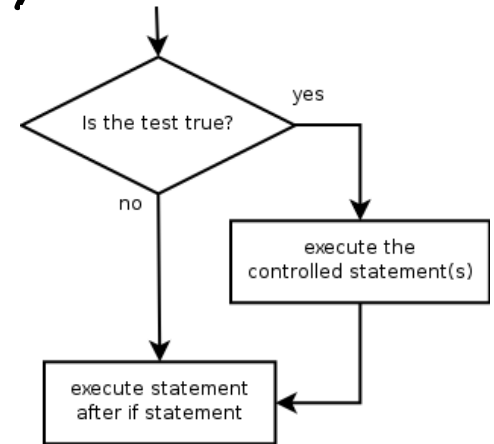
❑ Robust design

1. Add a **test** method to check whether the input has the expected type
2. **if** the test failed, report error

The `if` statement

Executes a block of statements only if a test is true

```
if (test) {  
    statement;  
    ...  
    statement;  
}
```



□ Example:

```
double gpa = console.nextDouble();  
if (gpa >= 3.9)  
    System.out.println("Welcome to XMU!");
```

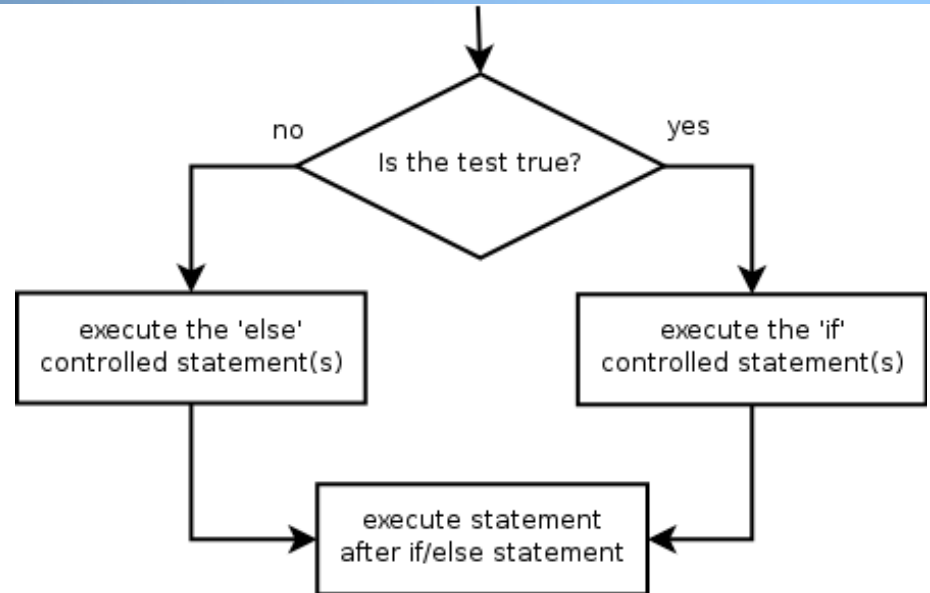
The if/else Statement

- An *else clause* can be added to an `if` statement to make it an *if-else statement*:

```
if ( test ) {  
    statement1;  
}  
else {  
    statement2;  
}
```

- If the condition is true, `statement1` is executed; if the condition is false, `statement2` is executed
- One or the other will be executed, but not both

The if/else Statement



□ Example:

```
if (gpa >= 3.8) {  
    System.out.println("Welcome to XMU!");  
} else {  
    System.out.println("We recommend Harvard.");  
}
```

Outline

- ❑ Admin and recap
- ❑ Text I/O
 - m Input: basic Scanner input
 - m Output: basic printf and String.format
- ❑ Program flow of control
 - m Boolean expressions

Token and Exception

- When a token is not the type that the scanner expects, since no reasonable (non-ambiguous) return value, Scanner **throws** an **exception** (panic)

```
System.out.print("Which year will you graduate? ");  
int year = console.nextInt();
```

Output:

```
Which year will you graduate? Timmy
```



```
java.util InputMismatchException
```

```
at java.util.Scanner.next(Unknown Source)  
at java.util.Scanner.nextInt(Unknown Source)  
...
```



Exceptions



- ❑ **Exception:** a programming language mechanism to represent a **runtime error**, e.g.,
 - dividing an integer by 0
 - trying to read the wrong type of value from a `Scanner`
 - trying to read a file that does not exist
- ❑ Java has defined a set of exceptions, each with a name, e.g., `ArithmeticException`, `InputMismatchException`, `FileNotFoundException`

Why Not a “Smarter” nextInt()

- ❑ For example, continue to scan the input to find the integer?
- ❑ Design principle: design of basic methods should **KISS (Keep It Simple and Stupid)**
- ❑ Higher level programs handle the case in their specific settings

Design Methodology: How to Handle Potential Exceptions?

- Two basic approaches
 - m Test before proceed
 - m Proceed and clean up (try/catch)

Robust Input Approach 1: Test Before Proceed

□ Design pattern

```
if ( <ExceptionConditionFalse> ) {  
    proceed;  
}  
else {  
    System.out.println("Error message.");  
}
```

return type of hasNextInt() is boolean,
indicating a logical condition.

```
System.out.print("Which year will you graduate? ");  
if ( console.hasNextInt() ) {  
    year = console.nextInt();  
}  
else {  
    System.out.println("Wrong type; please give int.");  
}
```

Robust Input Approach 2: try and catch

- ❑ **try/catch**: if an exception happens, program execution jumps to the `catch` statement, skipping the rest in the `try` block.

```
try {  
    potentially dangerous statements  
}  
catch (ExceptionName e) {  
    handle exception, such as print an error message  
}  
// end of catch
```

Robust Input Approach 2: Example

```
import java.util.Scanner;    // for Scanner

public class ScannerInputExampleTry {
    public static void main(String[] args) {
        Scanner console = new Scanner( System.in );
        try {
            System.out.print("Which year will you graduate? ");
            int year = console.nextInt();
            System.out.println("You give " + year);
        } catch (InputMismatchException e) {
            // print an error message
            System.out.println("You give a wrong input type.");
        } // end of catch

    } // end of main
}
```

File as Scanner Source

```
import java.io.File;           // for File
import java.util.Scanner;      // for Scanner

public class PlotFile {
    public static void main(String[] args) {
        File f = new File("USA.txt");
        Scanner input = new Scanner( f );
        ...
    } // end of main
}
```

❑ **Compilation fails with the following error:**

```
PlotFile.java:9: unreported exception java.io.FileNotFoundException;
must be caught or declared to be thrown
    Scanner input = new Scanner( f );
```

What Happened: the throws Clause

- ❑ **throws clause:** Keyword on a method's header to state that it may generate an exception (and will not handle it) and those using it **must handle** it (called a **checked** exception; `nextInt` does not declare it).

- ❑ **Syntax:**

```
public static <type> <name>(...) throws <type> {
```

- m **Example:**

```
http://docs.oracle.com/javase/7/docs/api/java/util/Scanner.html#Scanner\(java.io.File\)
```

Scanner File Input

```
import java.io.File;           // for File
import java.util.Scanner;      // for Scanner

public class PlotFile {
    public static void main(String[] args) {
        try {
            File f = new File("USA.txt");
            Scanner input = new Scanner( f );

            ...
        } catch (FileNotFoundException e) {
            // print an error message
            System.out.println("File not found exception");
        } // end of catch

    } // end of main
}
```

Outline

- ❑ Admin and recap
- ❑ Text I/O
 - m Input: Scanner input
 - Scanner using object to remember state
 - Scanner input with exceptions (run time errors)
 - m Output: basic printf and String.format

A Tiny Bit History of Java

Text Formatting

- ❑ Before Java 1.5, Java provides formatting classes such as `NumberFormat` and `DecimalFormat` classes as part of the `java.text` package
- ❑ But many programmers like the more flexible method signature of `printf()` starting from the C programming language
- ❑ Starting from Java 1.5, `printf/formatr` is added and typically preferred by many programmers

Discussion

- ❑ Text output formatting as of now
 - m String concatenation without ability to specify per variable format

Printf/Format Design

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

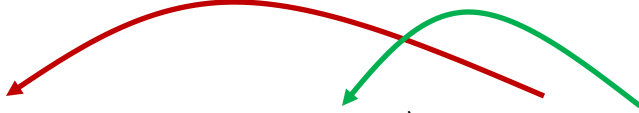
- Output with *placeholders* to insert parameters, e.g.,

m %d	integer
m %f	real number
m %s	string

- these placeholders are used instead of + concatenation

m 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/Format Design and Language Support

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

- In the most general case, Java allows flexible (**variable**) method signature, e.g.,

```
public static type name( <type1> param1,  
                        <type2>... param2)
```
- Number and type of parameters determined by the first parameter. We will not learn how to define such methods, but will use some: `printf()` and `format()`

printf Width

m **%Wd** integer, **W** characters wide, right-aligned
m **%-Wd** integer, **W** characters wide, left-aligned
m **%Wf** real number, **W** characters wide, right-aligned
m ...

```
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

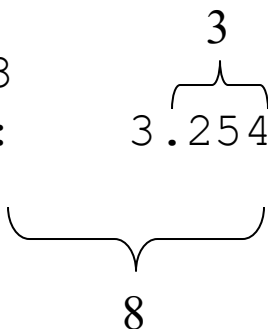
printf Precision

- `m %.Df` real number, rounded to **D** digits after decimal
- `m %W.Df` real number, **W** chars wide, **D** digits after decimal
- `m %-W.Df` 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
```



printf Formatting

- Many more formatting control options supported by `printf`, e.g., using the comma (,) to display numbers with thousands separator

```
System.out.printf("%,d\n", 58625);  
System.out.printf("%, .2f\n", 12345678.9);
```

Output:

```
58,625  
12,345,678.90
```

System.out.printf and String.format

- `String.format` has the same formatting capability as `printf`, except that `printf` outputs and `String.format` returns:

```
System.out.printf("%,.2f\n", 12345678.9);
```

```
String s = String.format("%,.2f\n", 12345678.9);  
System.out.print( s );
```

Output:

```
12,345,678.90
```

```
12,345,678.90
```

Exercise: F2C

- Revise F2C to print 2 decimal digits.

Foundational Programming Concepts

any program you might want to write

objects

methods and classes

graphics, sound, and image I/O

arrays

conditionals and loops

math

text I/O

primitive data types

assignment statements

Program Flow of Control

- ❑ Java has three types of program flow of control:
 - m decision statements, or **conditional statements**:
decide whether or not to execute a particular statement
 - m repetition statements, or **loop statements**:
perform a statement over and over repetitively
 - m exceptions: to handle run-time errors (atypical)
- ❑ The foundation of conditional/loop program flow of control is the logical condition, which should be **a boolean expression**

Basic Boolean Condition: Relational Comparison



- A basic Boolean expression is to compare two values using a *relational operator* :

Operator	Meaning	Example	Value
==	equals	1 + 1 == 2	true
!=	does not equal	3.2 != 2.5	true
<	less than	10 < 5	false
>	greater than	10 > 5	true
<=	less than or equal to	126 <= 100	false
>=	greater than or equal to	5.0 >= 5.0	true

- Note the difference between the equality operator (==) and the assignment operator (=)

Example: Basic Boolean Expression

```
public class Flip {  
    public static void main(String[] args) {  
        if (Math.random() < 0.5) System.out.println("Heads");  
        else System.out.println("Tails");  
    }  
}
```

Flip.java



```
% java Flip
```

```
Heads
```

```
% java Flip
```

```
Heads
```

```
% java Flip
```

```
Tails
```

```
% java Flip
```

```
Heads
```

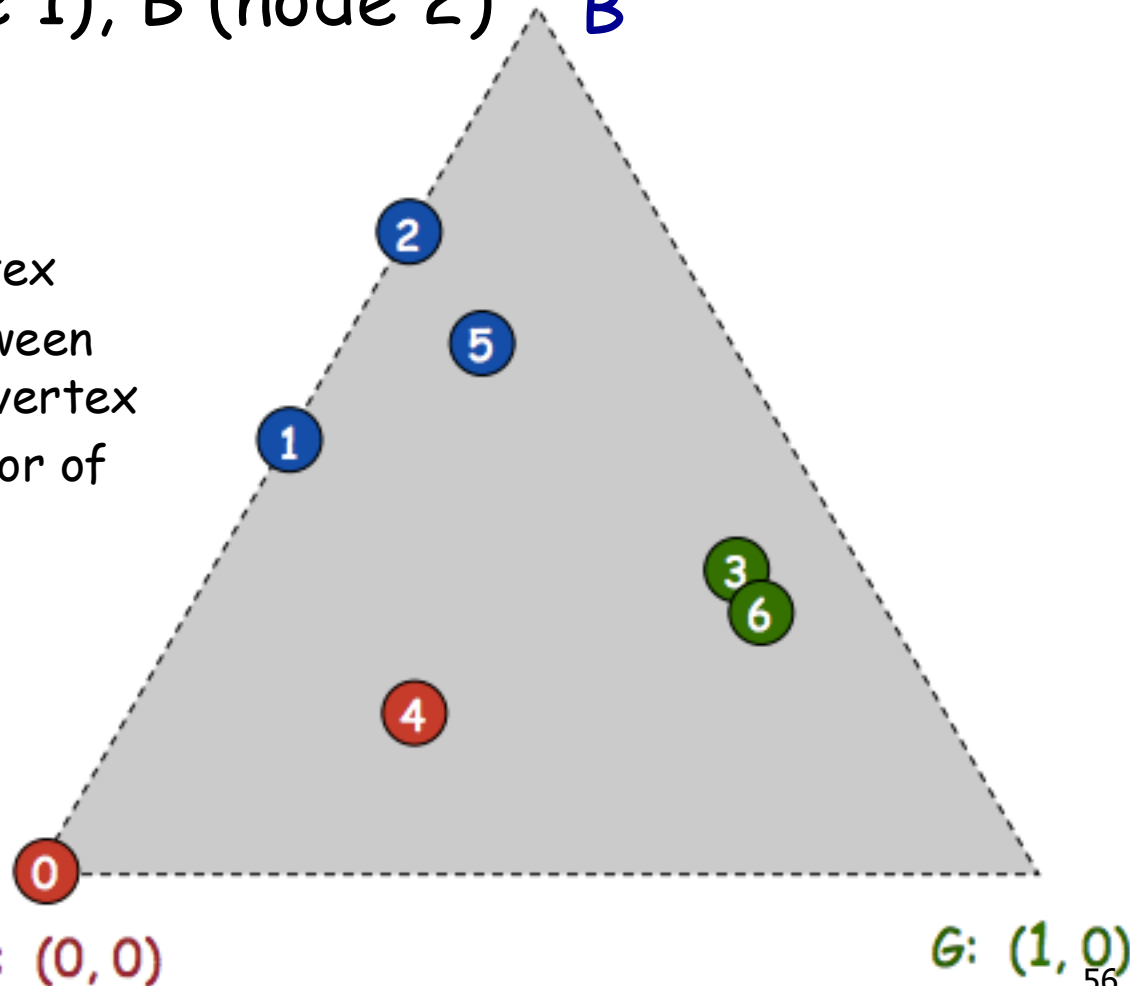
(Offline) Example: Chaos Game

□ Play on equilateral triangle, with vertices R (node 0), G (node 1), B (node 2)

m Start at R

m Repeat N times

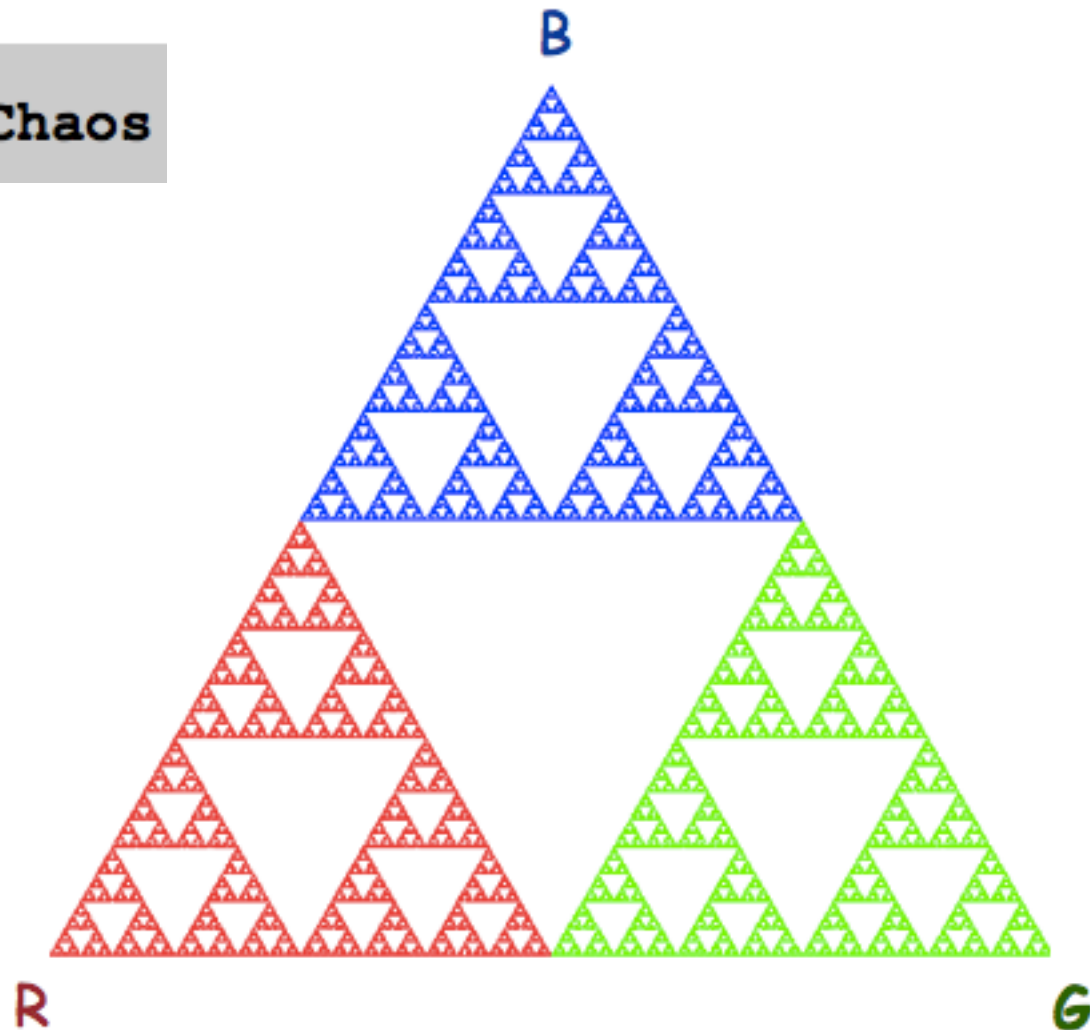
- Pick a random vertex
- Move halfway between current point and vertex
- Draw a point in color of chosen vertex



Chaos.java

(Offline) Example: Chaos Game

```
% java Chaos
```



Sierpinski triangle