Insights:

Programmers must be prepared to understand the language of a variety of application areas (mathematics, music, biology, civil engineering, art) so that they can discuss problems with domain experts.

One day you will switch to some other programming language; one of your first tasks will be figuring out its unit-testing framework.

Writing comments is imperative because programmers don't create programs for themselves. Programmers write programs for other programmers to read. They need to reread the programs from a month ago, a year ago, or twenty years ago and change them.

Programmers must decide how to use the chosen programming language to *represent* the relevant pieces of information as data and how we should *interpret* data as information.

A multi-function program should also come with a purpose statement. Indeed, good programmers write two purpose statements: one for the reader who may have to modify the code and another one for the person who wishes to use the program only.

TextBook: Read your butt off, worth every penny of it.

Recursion:	
Pros:	Cons:
Bridges the gap between elegance and complexity	Slowness due to CPU overhead
Reduces the need for complex loops and auxiliary data structures	Can lead to out of memory error stack overflow exceptions
Can reduce time complexity easily with memorization	Can be unnecessarily complex if poorly constructed

Works really well with recursive structures like lists, trees and graphs	
Structure:	
Define the base case stopping condition While reduce the size and call myself, do something about the current **unit** of the input.	after the base case, I know i have to call the same method/function, assume the return is correct, and working on the same level with the current value to return a correct final result
Call Stack:	
hey next, what's your value? hey next, he is bigger than me, please compare with him, when you guys are figured out, I'll link with whoever is smaller.	
Examples:	
Reverse a String: base case is empty string; body is call self with shrank input + the current element.	*String is Palindrome or not*: base case is empty or one char; if (first and last is the same): self call on shrank input return false
On numbers: *Decimal to Binary;*	
Divide and Conquer:	
normally recursive; • divide problem into several smaller subproblems; conquer the subproblems by solving them recursively.	• base case: solve small enough problems by brute force • combine the solutions all the way back to the original problem
Binary Search on a list	Fibonacci (using memorization)
merge-sort of a list	linked list reverse
Merge two sorted linked list	

Insert value into binary search tree	print all leaf node on a tree (Depth first with recursion)
Depth first search on graph (difference on trees is that it could be a circle, so need an accumulator to track the visited nodes)	

Tree:	
Binary Tree: - at most 2 children per node - exactly 1 root - exactly 1 path between root and any node	- root has no parent - leaf has no children
Coding-wise: - each node is a object - a node holds a value, left child, right childr	
Breath-first search: - iterative approach with a stack - recursive (stack order by nature)	Depth-first search: - only iterative approach with a queue
recursive solution: - draw the tree out - empty tree and leafs would be the base cases - get the return (boolean, sum of ints etc) from left and right do something with the current	

Java:

```
javac  BinarySearch.java
java BinarySearch
```

A Java program (class) is a data-type definition;
Like C, could also be a library of static methods(functions).

Data in Java:

Data types:

Primitive (eight): boolean / byte / short / char / int / float / long / double
User-defined: Any thing the program(classes) defines.

Data-type forms:

- A data-type(class) of primitive data types as its field.

- A data-type (class) contains another data-type (class) as its field.

- Union data:
  Similar data-types union under an Interface / Abstract class
  - keywords: “interface” / “implements”.
  - keywords: “abstract class” / “extends”.
    (If the subclasses share the same fields)

- Recursive Data:
  Use union and the first case is the base case.

```
interface IAT{ }
class unknown implements IAT { }
class Person implements IAT {
      String name;
      IAT mom;
      IAT dad;
      Person(String name, IAT mom, IAT dad) {
            this.name = name;
            this.mom = mom;
            this.dad = dad;
      }
}
```

More details on Data-type forms:

– – – – – – – – – – – – – – – – – – – –

Union data-type:

Interfaces:

Every class that implements the interface is required to implement all methods that the interface specifies. In turn, whenever we use any object of the type specified by the interface, we are guaranteed that we can invoke every method defined in the interface on this object.

It'a tag to describe a group of classes are related.

It describes a list of things we can do to the related date types in a useful way.

Abstract Class:

An important principle in program design: Don't repeat yourself.

We say the method definition in a concrete class **overrides** the definition in its superclass. At runtime, Java looks for a method with the matching signature (matching header) first in the class where the current instance of the object has been defined. If the method is not found, it continues looking in its superclass.

Abstract case 1:

When two or more fields could be a data-type (e.g. class CarPt for class Circle and class Square); int x, int y are common fields, and methods on them are repetitive.

Abstract case 2:

If two classes have the common fields, try to abstract them out into an abstract class. (e.g. abstract class AShape for class Circle and class Square).

Constructor:

super() for base classes, this() for within the superclasses.

Check validity and throw new IllegalArgumentException();

Utility class:

When code truly is a *function* — it makes no mention of **this**, and really doesn't depend on any object at all. Utility class acts as a container for

these functions that don't have any other class they really need to be part of.

Function object:
Abstract over behavior;
interface IPredicate { }
interface IComparator { } // need helper

Methods:

Represent the behavior of the objects.

Delegation:
A data-type is a field of another data-type: Avoiding field of field access. When a method in this object needs to access other object's fields, it delegates the task to "that object" by invoking a method in that object.

Dynamic dispatch:
For union data-type.
It is one of the cornerstone concepts of object-oriented programming.
Dispatching to the correct method implementation based on information available only dynamically. (e.g. someObject **.** someMethod(…);)
At compile-time, the " . " asks hey what object are you (out of many classes of data-types). Then, Okay here, this is your Method.

Can return primitive types or objects itself.

Data structures:

We've now seen several data structures that can be used to store collections of items: ILists, ArrayLists, BinaryTrees and binary search trees, and Deques. Primarily we have introduced each of these types to study some new patterns of object-oriented

construction: interfaces, linked lists, indexed data structures, branching structures, and wrappers and sentinels.
List:
(Using union-recursive data-type)

```
interface ILoBook { }
class MtLoBook implements ILoBook { }
class ConsLoBook implements ILoBook {
        Book first;
        ILoBook rest;

        ConsLoBook(Book first, ILoBook rest) {
                this.first = first;
                this.rest = rest;
        }
}
```

Common Methods on Lists:
Filtering:
If (delegation is true) put it in the list; else checking next one.

Sorting:
this.rest.sort().insert(this.first).

Common methods on Trees:
Helper: When excluding “this”, use helper to let dynamic dispatch decide to go to the base case or go to the helper.

Helper with an accumulator

Electricity between towns:
The lack of backup connections implies that a solution must be a tree. (As opposed to a graph that might have multiple paths or cycles).

The connectedness requirement means it should be a spanning tree. (As opposed to a disconnected forest of multiple trees).

The cost focus means it should be a minimum spanning tree.
(As opposed to a more expensive tree).

Run time:

Constant time / O(1):
when the input size doesn't matter, for example, getFirst(data of size n).

Logarithmic time / O(log n):
When it reduces the size of the input data in each step (it don't need to look at all values of the input data). e.g. binary search; operations on binary trees;
Linear time / O(n):
When the running time increases at most linearly with the size of the input data. This is the best possible time complexity when the algorithm must examine all values in the input data.

Quasilinear time / o(n log n):
When each operation in the input data have a logarithm time complexity. It is commonly seen in sorting algorithms (merge sort, heap sort …).

Quadratic time / O(n^2):
When it needs to perform a linear time operation for each value in the input data. e.g. for x in data { for y in data { print(x, y); }} ; bubble sort.

Exponential time O(2^n):
When the growth doubles with each addition to the input data set. It's usually seen in brute-force algorithms. Another example is recursive calculation of Fibonacci numbers:

```
int fibonacci(n) {
        if (n <= 1) { return n; }
        else { return fionacci(n - 1) + fibonacci(n - 2); } }
```

Factorial time / O (n!):
When it grows in a factorial way base on the size of the input data.

```
2! = 2 x 1 = 2
3! = 3 x 2 x 1 = 6
4! = 4 x 3 x 2 x 1 = 24
5! = 5 x 4 x 3 x 2 x 1 = 120
```

```
6! = 6 x 5 x 4 x 3 x 2 x 1 = 720
7! = 7 x 6 x 5 x 4 x 3 x 2 x 1 = 5.040
8! = 8 x 7 x 6 x 5 x 4 x 3 x 2 x 1 = 40.320
```

A great example of an factorial time complexity algorithm is the Heap's algorithm, which is used for generating all possible permutations of n objects.

Memory Space:

Instead of measuring exact memory usage, we should count how many objects are created.

Tips:

Java variables cannot hold objects directly - objects are compound data structures - so instead variables must hold a reference to an object.

When making a copy of a list, if don't want to copy the reference, only way is use for loop copy each element into the new list.

When overriding equals, has to use instanceof.

Because some cells may be empty, we need some way to distinguish empty cells from player numbers. The wrapped integer type Integer includes null, so we use Integer instead of int for the result with null representing empty cells.

Sameness:
Intensional / physical : (==) check if two things refer to the same object.
Extensional / logical : check if things the object represents are the same.

always test whether two **String**s are equal by their equals method;
always check whether two **int**s are equal by using the == operator.

t.checkExpect(4.3333 == 4.3333, true) // BAD idea
t.checkExpect(4.3333 - 4.3333 < 0.001, true) // MUCH BETTER
t.checkInexact(4.3333, 4.3333, 0.001);

// FAILS: the absolute difference is much bigger than 0.1
t.checkExpect(1100000 - 1000000 < 0.1, true);

// SUCCEEDS: the relative difference is at most 10%
t.checkInexact(1100000, 1000000, 0.1);

Instance of and Cast (not good when there is a subclass, Rect - Square).

Using double-dispatch to test for sameness

if (r1.time < r2.time) return -1;
else if (r1.time == r2.time) return 0; VS. r1.time - r2.time
else return -1;

Double.POSITIVE_INFINITY
Double.NEGTIVE_INFINITY

IAT : unk = new Unknown( );
IAT : jeni = new Person("Jeni", this.unk, this.unk);
IAT : carl = new Person("Carl", this.jeni, this.dan);
the current value of this.dan is null; and the initialization next won't change the null to its value.
IAT : dan = new Person("Dan", this,unk, this.unk);

```
While (!StdIn.isEmpty()) {

    String item = StdIn.readString();
}

int[] whitelist = In.readInts(args[0]);
```

Sets = unordered collections of elements without duplicates.
Bags = unordered collections of elements with duplicates.
Tables = associations of keys and values.

C:

```
$ gcc hello.c
$ ./a.out
Hello, World!

$ gcc hello.c -o hello
$ ./hello
Hello, World!
```

A C-program is a collection of one or more functions:

- **Library functions: printf( ), scanf( )…**
- **User-defined functions: anything( )…**

- Only one function: must be **main( );**
- More than one function: one and only one must be main( );
- Program execution always begins with main( );
- Other functions are called in sequence specified by function calls in main( );
- Any function can be called in any other function, even main( ) can be called in other functions; but a function cannot be defined in another function; a function can call itself (recursion).
- A function can be called any number of times;
- The order in which the functions are defined in a program and the order in which they get called are NOT necessarily the same.

- If a called function has no meaningful return-value, the moment closing brace "}" is encountered, the control returns to the calling function. Otherwise, a return statement is needed.
- After each function has done its thing, control returns to main( );
- When main( ) runs out of function calls, the program ends.
- #include <stdio.h>: Before we call the library functions we must write their prototype. This helps the compiler checking whether the values being passed and returned are as per the prototype declaration. But since we may not know the prototypes, ".h" files contain the prototypes of library functions. e.g: prototypes of all I/O functions are provided in the file "stdio.h"; prototype of math functions are provided in "math.h".
- Any C function by default returns an int value; If we desire that a function should return a value other than an int, then it is necessary to explicitly mention so in the calling function as well as in the called function.
- & means "address of" : (if "int a = 3"; "&a" is the location number that computer stores 3).
- * means "value at address" : int *j; j = &a. int **k means k is a pointer to an integer pointer.
- Pointer variables: a variable contains address of another variable.
  - int *j means the value at the address contained in j is an int.
  - char *ch means the value at the address contained in ch is a char.
- "macro" vs "functions"

change scanf("%c", &c); to scanf(" %c", &c);
When reading your input: except %d, %c takes enter as the input. When you hit enter! Enter is a character! By putting a space before %c it will consume enter, and take you input to %c.

Difference:

– – – – – – – – – – – –
C:
Functional Programming Language that has two types of functions:
Library functions printf( ), scanf( ), etc.
User-defined functions.

Pointer:
Pointers are variables which hold addresses of other variables. Return more than one value simultaneously.
&i “address of i”.
*j “value at address contained in j”.

C can declare a variable in a file.
C: #include <stdio.h>;
Java: import StdIn;
C’s default return type is int;

Packages:
package lessons