

CS2020

Data Structures and Algorithms

Welcome!

Announcements

Join CS2020 on:

- Coursemology
- Nota Bene
- Facebook

Beware:

- Everything was reset on Thursday.
- If you registered before Thursday, go do it again!

Register on CORS for:

- Tutorials (Discussion Groups)
- Recitations (Problem Sessions)

Jokes?

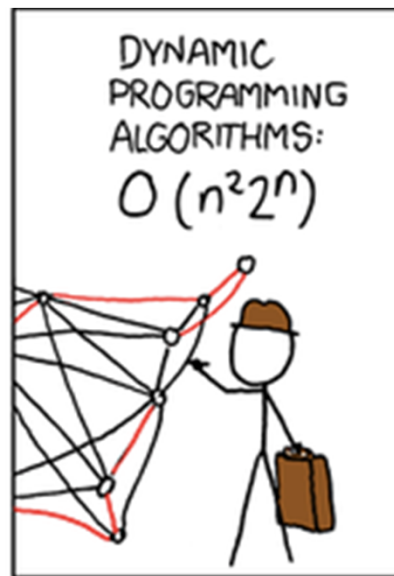
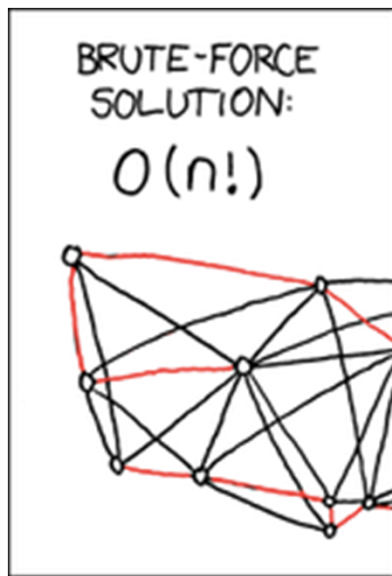
A train station is where a train stops. A bus station is where a bus stops. And on my desk, I have a workstation.

A co-mathematician is a device for turning co-theorems into ffee.

Why did the functions stop calling each other? Because they had constant arguments

Why did the programmer confuse Halloween and Christmas?
Because Oct 31 == Dec 25

Once upon a time, there were three kingdoms, all bordering on the same lake. For centuries, these kingdoms had fought over an island in the middle of that lake. One day, they decided to have it out, once and for all.



XKCD



Announcements

If you are still not registered for CS2020:

- Today is the last day to add a module.
- Talk to me ASAP.
- For those who signed up on Wednesday:
 - You should have seen an e-mail from me.
 - Please follow up with any questions they may ask.

Announcements

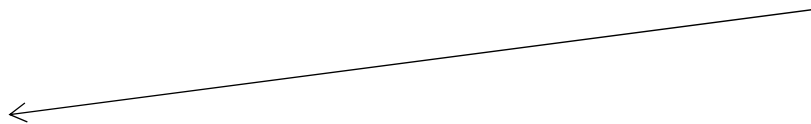
Problem Sessions:

- Come today to any of the three sessions!

- Today: COM1-208

- 1pm – 2pm
- 2pm – 3pm
- 3pm – 4pm

Pro tip: choose this one!



- Choose any one!

Announcements

Problem Set 1:

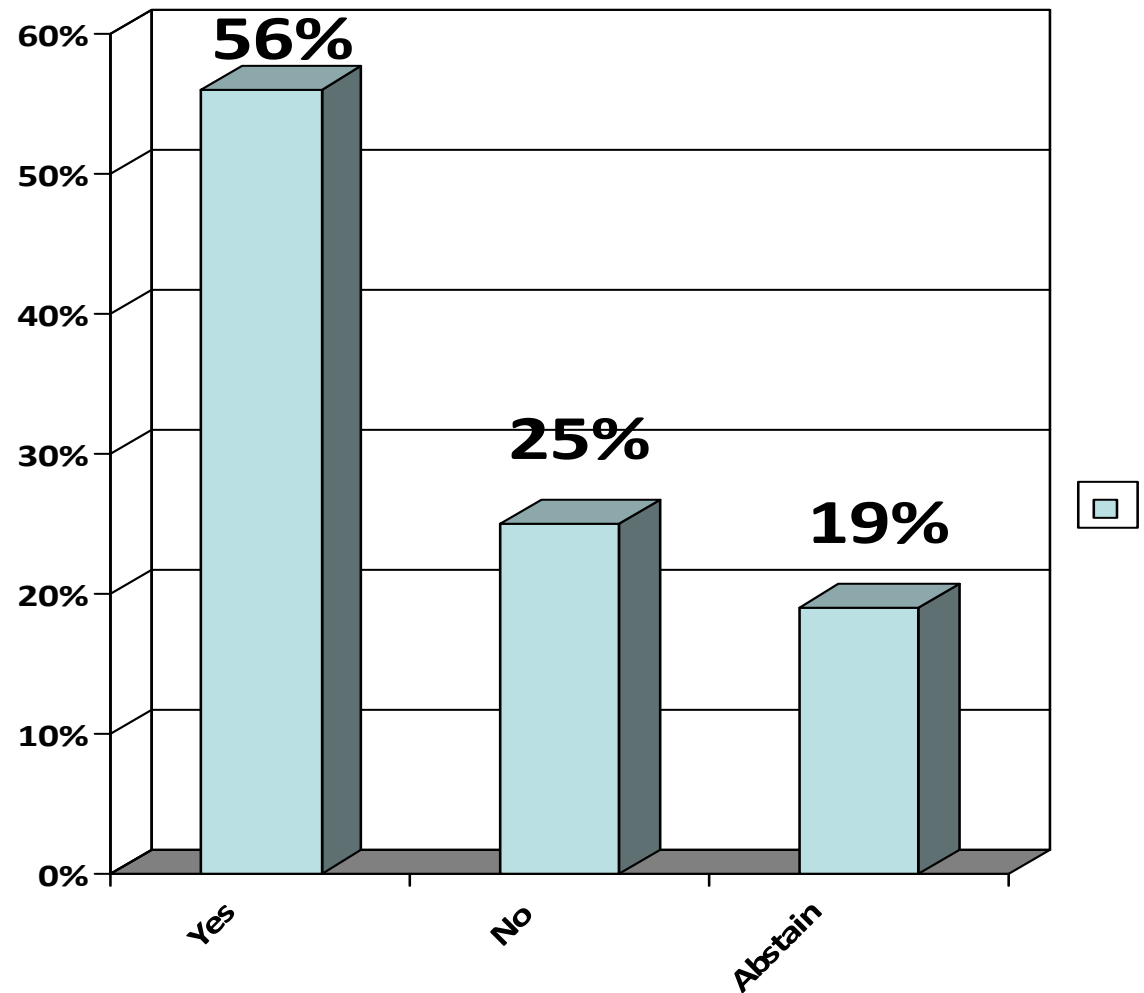
- Available on Coursemology
- Due next Tuesday night
- Discussion on NB (Nota Bene)

Beware:

- There was a “bug” in the “secret code” for decoding the image.
- It was fixed on Thursday morning.

Did you remember to bring your clicker?

- ✓ a. Yes
- b. No
- c. Abstain



Bring your clickers to class.

Today

Java Overview

Object-oriented Programming

Programming Paradigms

Models of programming:

- Procedural (imperative) languages
- Functional languages
- Declarative languages
- Object-oriented languages

How to organize information?

How to think about a solution?

Programming Paradigms

Procedural Languages

- Examples:

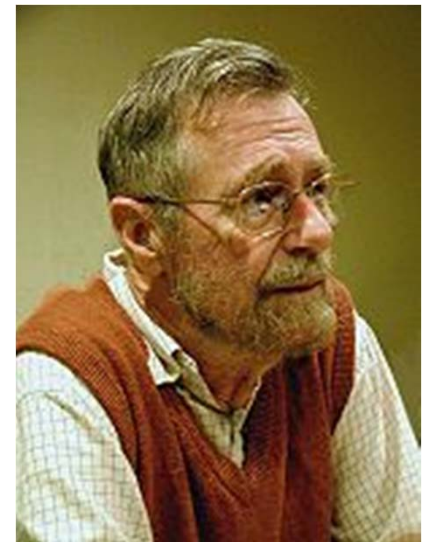
- Algol60, Fortran, COBOL, BASIC, Pascal, C

- Organization:

- Group instructions into “procedures” or “functions”
- Each procedure modifies the **state**.
- Don't use GOTO statement (see...)

- Advantages:

- Readability
- Procedure re-use



Programming Paradigms

Procedural Languages



- Each procedure modifies the state.
- Don't use GOTO statement (see...)

– Advantages:

- Readability
- Procedure re-use



Programming Paradigms

Functional Languages

- Examples:

- Scheme, Lisp

- Organization:

- Everything is a function
- Output depends only on input
- No state, no mutable data

- Advantages:

- Simplicity, elegance
- Describe what you are doing with *verbs*.
- Focus on computation, not data manipulation

Programming Paradigms

Object-oriented Languages

- Examples:

- Java, C++

- Advantages:

- Near-ubiquitous in industry
- Modular
- Code re-use
- Easier to iterate / develop new versions
 - Information hiding
 - Pluggable

Object-oriented Paradigm

Encapsulation

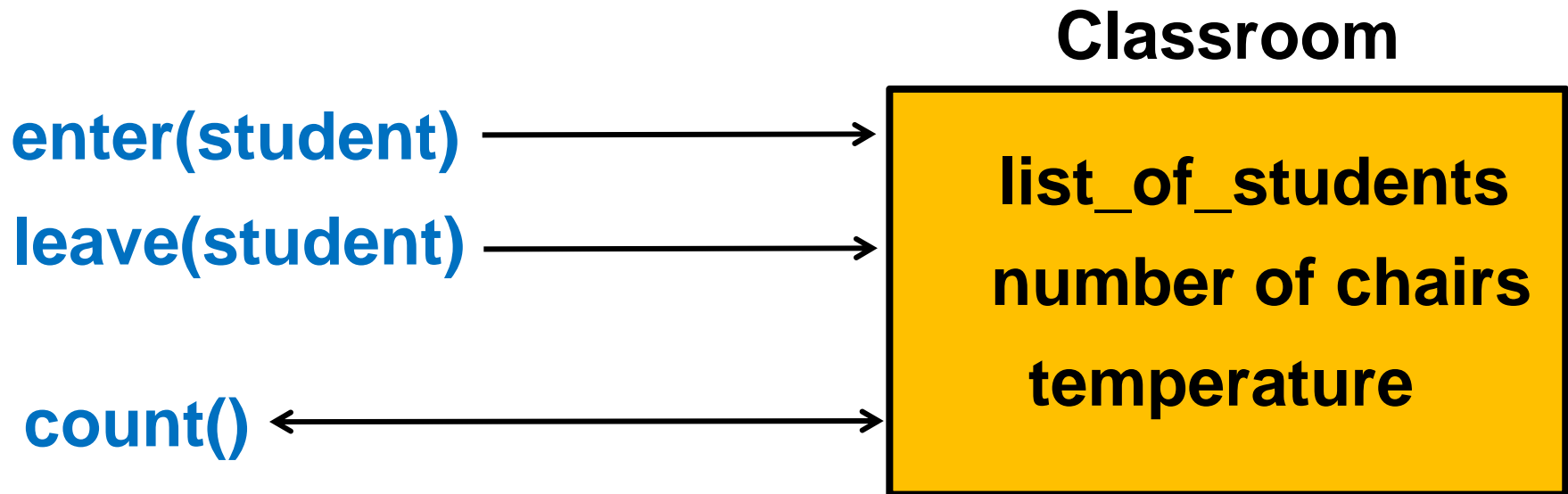
Inheritance

Polymorphism

Object-oriented Programming

Object has:

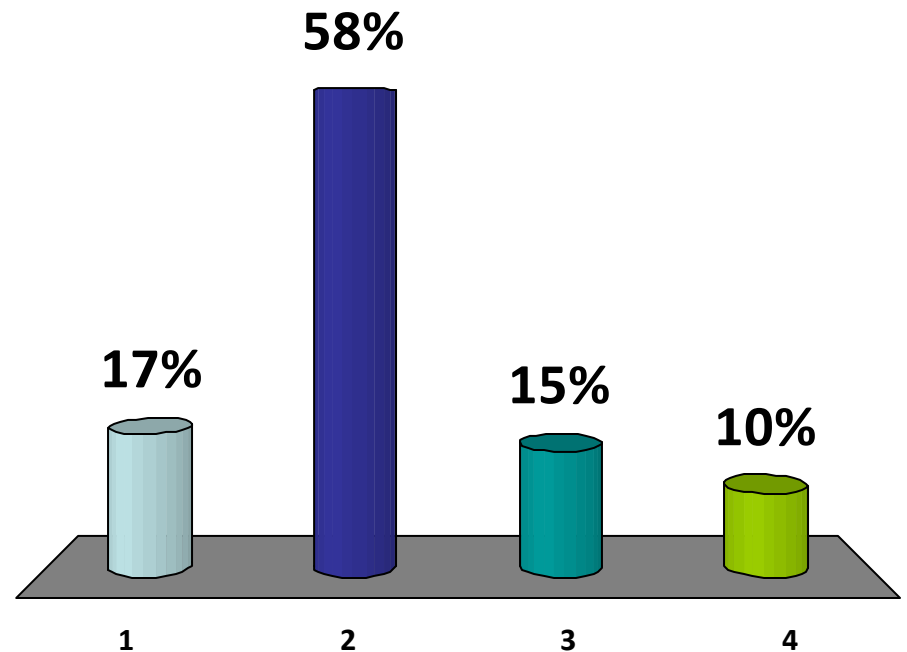
- State (i.e., data)
- Behavior (i.e., methods for modifying the state)



How to implement a **File System**?

How to implement a file system?

1. file management object + file contents object
- ✓ 2. file object + folder object
3. folder hierarchy + folder contents
4. file object



How to implement a **File System**?

Files:

- Contain data
- Edited
- Rename
- Moved

Folders:

- Contain files
- Contain folders
- Rename
- Moved

First principle of Java

« Everything is an object »

Defining a class in Java

```
class File
{
    String m_name = "";

    FileData m_contents = null;

    void rename(String newName) {...}

    FileData getData() {...}

    void setData(FileData newdata) {...}
}
```

Defining a class in Java

```
class File
```

```
{
```

```
    String m_name = "";
```

```
    FileData m_contents = null;
```

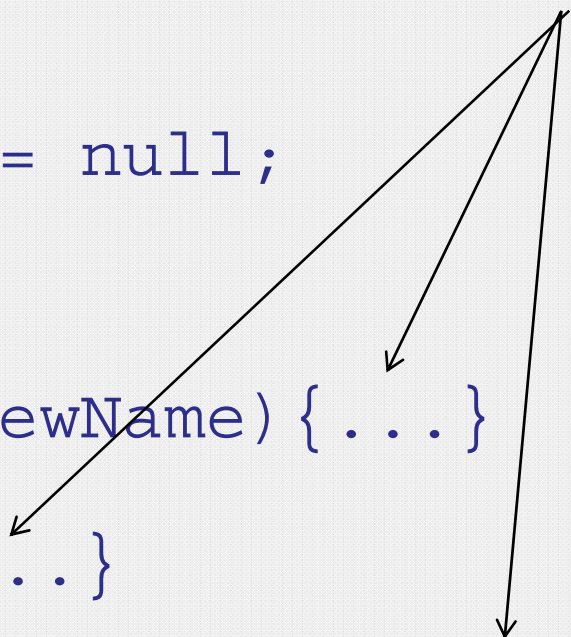
```
    void rename(String newName) {...}
```

```
    FileData getData() {...}
```

```
    void setData(FileData newdata) {...}
```

```
}
```

*Not allowed in Java.
Abbreviation in
Powerpoint for
missing code.*



Defining a class in Java

```
class File
{
    String m_name = "";

    FileData m_contents = null;

    void rename(String newName) {...}

    FileData getData() {...}

    void setData(FileData newdata) {...}
}
```

Defining a class in Java

class File

```
{  
    String m_name = "";  
    FileData m_contents = null;  
  
    void rename(String newName) {...}  
    FileData getData() {...}  
    void setData(FileData newdata) {...}  
}
```

Defining a class in Java

```
class File
{
    String m_name = "";

    FileData m_contents = null;

    void rename(String newName) {...}

    FileData getData() {...}

    void setData(FileData newdata) {...}
}
```


Types

```
class File
{
    String m_name = "";
    FileData m_contents = null;

    void rename(String newName) {...}
    FileData getData() {...}
    void setData(FileData newdata) {...}
}
```

Defining a class in Java

```
class File
```

```
{
```

```
    String m_name = "";
```

```
    FileData m_contents = null;
```

```
    void rename(String newName){...}
```

```
    FileData getData(){...}
```

```
    void setData(FileData newdata){...}
```

```
}
```

Member (Instance) Variables

```
class File
{
    String m_name = "";
    FileData m_contents = null;

    void rename(String newName) {...}
    FileData getData() {...}
    void setData(FileData newdata) {...}
}
```

Defining a class in Java

```
class File
```

```
{
```

```
    String m_name = "";
```

```
    FileData m_contents = null;
```

```
    void rename(String newName){...}
```

```
    FileData getData(){...}
```

```
    void setData(FileData newdata){...}
```

```
}
```


Methods (Functions)

```
class File
{
    String m_name = "";
    FileData m_contents = null;

    void rename(String newName) {...}
    FileData getData() {...}
    void setData(FileData newdata) {...}
}
```


Defining a class in Java

```
class File
{
    String m_name = "";

    FileData m_contents = null;

    void rename(String newName) {...}

    FileData getData() {...}

    void setData(FileData newdata) {...}
}
```

Another class

```
class Folder
{
    String m_name;
    Folder[] m_children;
    File[] m_files;

    int getNumFiles() {...}
    File getFile(int i) {...}

}
```


Another class

```
class Folder
```

```
{
```

```
    String m_name;
```

```
    Folder[] m_children;
```

```
    File[] m_files;
```

```
    int getNumFiles() {...}
```

```
    File getFile(int i) {...}
```

```
}
```

Note array notation



Note “recursive” folder



Class vs. Object

What's the difference?

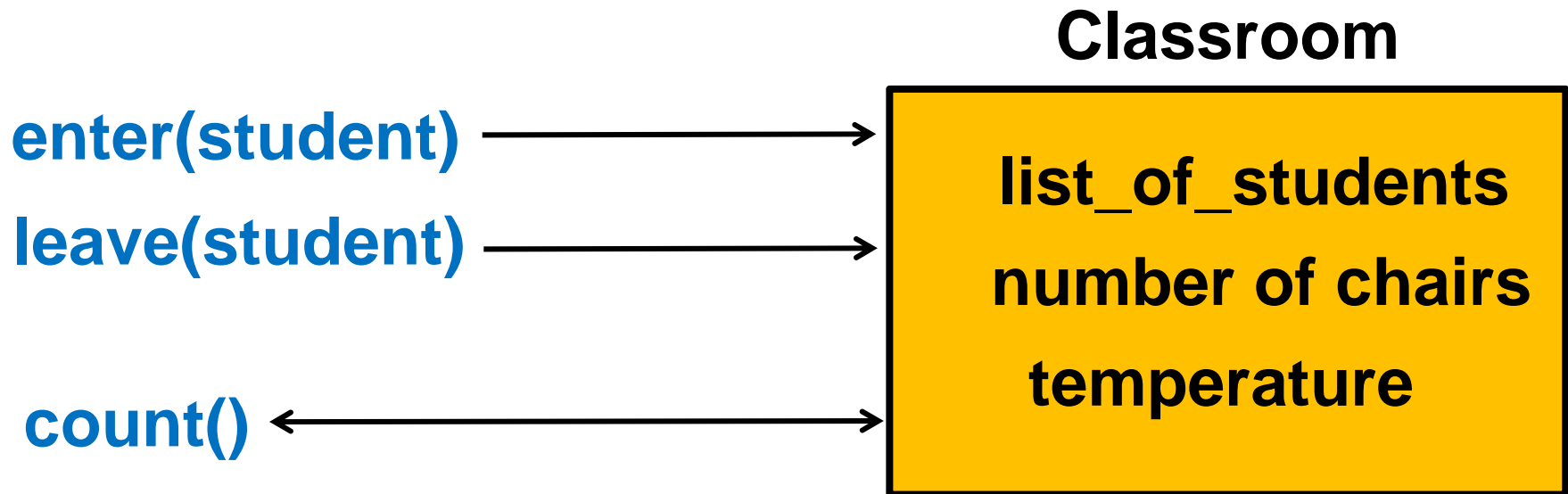
Creating a new object

```
Folder createFolder(String name) {  
  
    Folder redFolder = new  
    Folder(name);  
  
    return redFolder;  
  
}
```

Object-oriented Programming

Object has:

- State (i.e., data)
- Behavior (i.e., methods for modifying the state)



Object-oriented Paradigm

Encapsulation

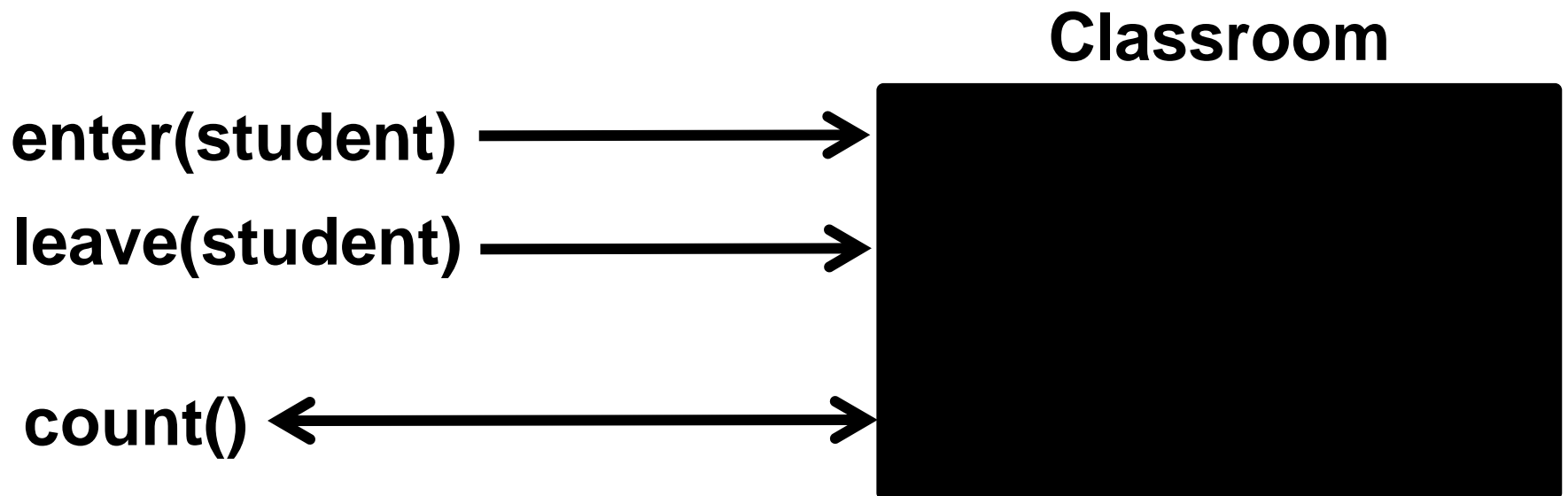
Inheritance

Polymorphism

Encapsulation

Interface: how you manipulate the object

Implementation: details hidden inside the object



Encapsulation

```
class File
{
    String m_name = "";
    FileData m_contents = null;

    void rename(String newName) {...}
    FileData getData() {...}
    void setData(FileData newdata) {...}
}
```

Defining an interface

```
// Comments go here
interface IFile
{
    // Comments explain how to use interface
    void rename(String newName);

    FileData getData();

    void setData(FileData newdata);
}
```

*Note no functionality!
Only method names.*

Implementing an interface

```
class File implements IFile
```

```
{
```

```
    String m_name = "";
```

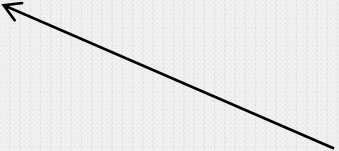
```
    FileData m_contents = null;
```

```
    void rename(String newName) {...}
```

```
    FileData getData() {...}
```

```
    void setData(FileData newdata) {...}
```

```
}
```



*"I promise to implement
all the functionality in IFile."*

Implementing an interface

```
class OtherFile implements IFile
{
    char[] nom;

    char[] meteo;

    FileData getData() {...}

    void setData(FileData nouveau) {...}
}
```

Implementing an interface

```
class OtherFile implements IFile  
{
```

```
    char[] nom;
```

```
    char[] meteo;
```

Error!

```
    FileData getData() {...}
```

```
    void setData(FileData nouveau) {...}
```

```
}
```

Implementing an interface

```
class OtherFile implements IFile
{
    char[] nom;

    char[] meteo;

    void rename() {...}

    FileData getData() {...}

    void setData(FileData nouveau) {...}
}
```


Using an interface

```
IFile copyFile(IFile oldFile) {
```

```
    File newFile = new File();
```

```
    FileData data = oldFile.getData();
```

```
    newFile.setData(data);
```

```
    return newFile;
```

```
}
```

It does not matter how the object is implemented. The oldFile can be a File or an OtherFile.

Quick summary...

So far:

- Defining classes and interfaces
- Implementing interfaces
- Using interfaces

Next:

- Access control
- Static variables / methods
- Initializing an object / Constructors

Access Control:

« Behavior is public, data is private »

Defining a class in Java

```
public class OtherFile implements IFile
{
    private char[] name;

    private char[] contents;

    public void rename() {...}
    public FileData getData() {...}
    public void setData(FileData newdata) {...}
    private void compressDataStorage()

}
```

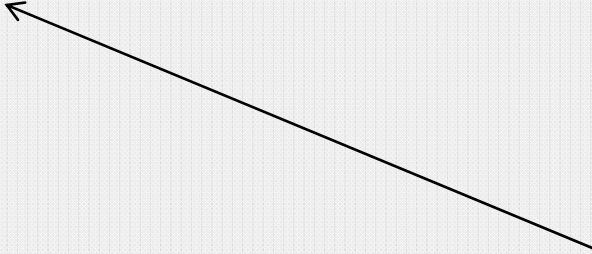
Access Control

- (none specified)
 - within the same package
- public
 - everywhere
- private:
 - only in the same class
- protected:
 - within the same package, and by subclasses

Access Control

```
public class A
{
    private int secretVariable;
}

public class B
{
    public int stealSecrets(A example){
        int readMe = example.secretVariable;
    }
}
```

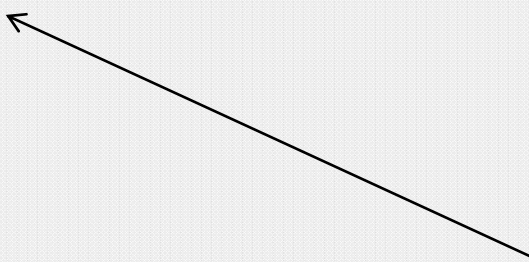


Error: cannot access secretVariable.

Access Control

```
public class A
{
    public int secretVariable;
}

public class B
{
    public int stealSecrets(A example){
        int readMe = example.secretVariable;
    }
}
```

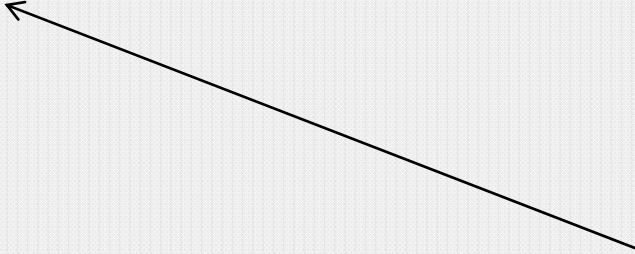


Ok, can access secretVariable.

Access Control

```
public class A
{
    private int secretFunction();
}

public class B
{
    public int stealSecrets(A example){
        int readMe = example.secretFunction();
    }
}
```



Error: cannot access secretFunction().

Access Control

- (none specified)
 - within the same package
- public
 - everywhere
- private:
 - only in the same class
- protected:
 - within the same package, and by subclasses

Packages

```
package com.mycompany.joe;
```

```
public class B  
{  
    public int stealSecrets(A example) {  
        int readMe = example.secretFunction();  
    }  
}
```

Organize code into packages. Every class should be in some package.

Package names should be unique.

Standard practice: reverse URL.

Packages

```
package com.mycompany.joe;
```

```
import cs2020.*;
```

Import everything from package cs2020.

```
public class B  
{
```

```
    public int stealSecrets(A example) {
```

```
        int readMe = example.secretFunction();
```

```
    }
```

```
}
```

Packages

```
package com.mycompany.joe;

import cs2020.ShiftRegister;

public class B
{
    public int stealSecrets(A example) {


        ShiftRegister reg = A.getRegister();

    }
}
```

Import just ShiftRegister from package cs2020.

Good practice: only import what you need.

Access Control

- (none specified)  *Don't do this!*
 - within the same package
- public
 - everywhere
- private:
 - only in the same class
- protected:
 - within the same package, and by subclasses

Class vs. Object

What's the difference?

static methods

```
class File
{
    private String m_name = "";
    private FileData m_contents = null;

    public static String addExt(String name) {
        return (name + ".pdf");
    }
}
```


static methods

```
class File
{
    private String m_name = "";
    private FileData m_contents = null;

    public static String addExt(String name) {
        m_fileName = name;
        return (name + ".pdf");
    }
}
```

static methods

```
class File
{
    private String m_name = "";
    private FileData m_contents = null;

    public static String addExt(String name) {
        m_fileName = name;
        return (name + ".pdf");
    }
}
```

Error!

Cannot access member variable.

static methods

```
class File
{
    private String m_name = "";

    private static int s_count = 0;

    public void increment() {
        s_count++;
    }
}
```

Every File object shares s_count.

Initializing an object

Initializing an object

```
class File
{
    private String m_name = "";
    private FileData m_contents = null;

    public File(String fileName) {
        m_name = fileName;
        m_contents = null;
    }
}
```

Initializing an object

```
class File
{
    private String m_name = "";
    private FileData m_contents = null;

    // Constructor
    public File(String fileName) {
        m_name = fileName;
        m_contents = null;
    }
}
```


Initializing an object

```
class File
{
    public File(String fileName) {
        m_name = fileName;
        m_contents = null;
    }

    public File() {
        m_name = null;
        m_contents = null;
    }
}
```

Multiple constructors with different signatures.

Initializing an object with an array

```
class File
{
    private int[] m_pageNumbers = new int[100];
}
```

If the array size is fixed, then initialization is simple.

What if the array size is not known in advance?

Initializing an object with an array

```
class File
{
    private int[] m_pageNumbers = null;

    public File(int NumPages) {
        m_pageNumbers = new int[numPages];
    }
}
```

You might use a constructor to initialize the array.

The main method

```
class FileSystem
{
    public static void main(String[] args) {
        Folder root = new Folder();
        File homework = new File("hw-one.txt");
        root.addfile(homework);
    }
}
```

Creating an object

```
class FileSystem
{
    public static void main(String[] args) {

        Folder root = new Folder();

        File homework = new File("hw-one.txt");

        root.addfile(homework);

    }
}
```

Using a constructor

```
class FileSystem
{
    public static void main(String[] args) {
        Folder root = new Folder();

        File homework = new File("hw-one.txt");

        root.addfile(homework);
    }
}
```


Invoking a method

```
class FileSystem
{
    public static void main(String[] args) {
        Folder root = new Folder();
        File homework = new File("hw-one.txt");
        root.addFile(homework);
    }
}
```

Java Operators

Operator	Functionality
=	assignment
+, -, *, /	plus, minus, multiplication, division
%	remainder
++, --	increment, decrement
==, !=	test equality
<, >	less than, greater than
<=, >=	less-than-or-equal, greater-than-or-equal
<<, >>	left shift, right shift
&&,	logical and, logical or
~, &, ^,	bitwise operations: complement, and, xor, or

Primitive Data Types

Name	Size	Min	Max
byte	8 bit	-128	127
short	16 bit	-32,768	32,767
int	32 bit	-2,147,483,648	2,147,483,647
long	64 bit	-9,223,372,036,854,775,808	9,223,372,036,854,775,808
float	32 bit		
double	64 bit		
boolean	1 bit	false	true
char	16 bit (unicode)	\u0000 (0)	\uffff (65535)

Find out more:

Java basics:

<http://docs.oracle.com/javase/tutorial/java/nutsandbolts/>

Java object-oriented programming:

<http://docs.oracle.com/javase/tutorial/java/javaOO/index.html>

Find out more:

Java basics:

<http://docs.oracle.com/javase/tutorial/java/nutsandbolts/>

Java object-oriented programming:

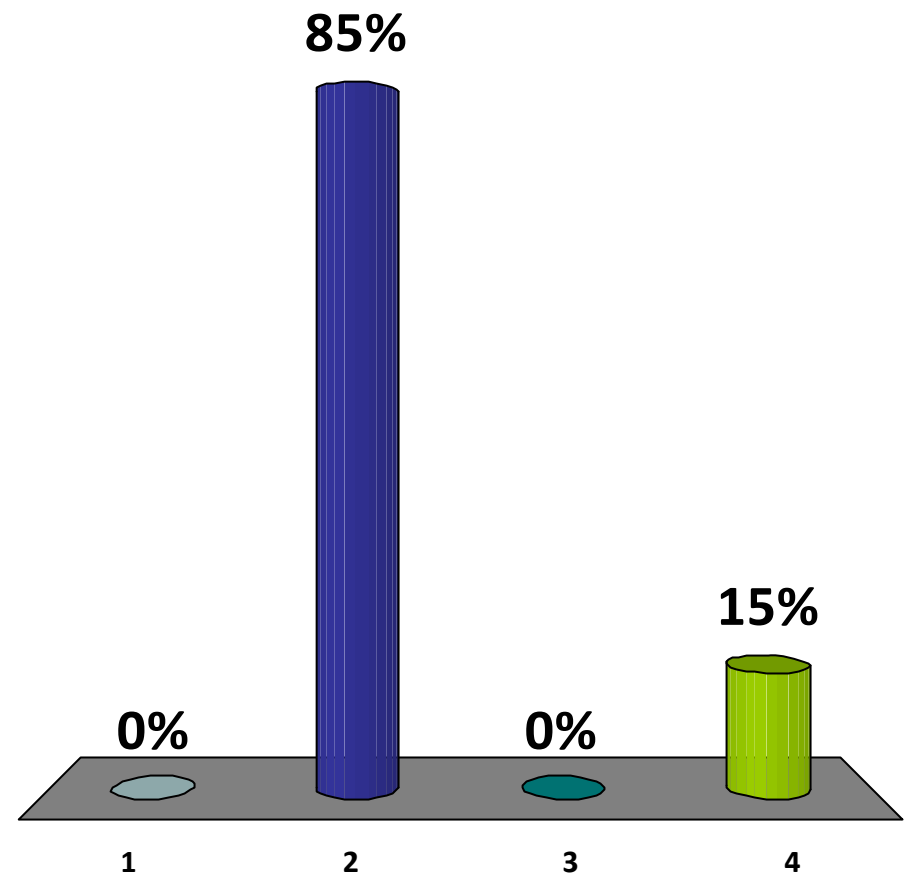
<http://docs.oracle.com/javase/tutorial/java/javaOO/index.html>

NB Assignment 2:

Go read / skim these documents. Add annotations to this slide with links to the most important parts, or questions on parts you don't understand.

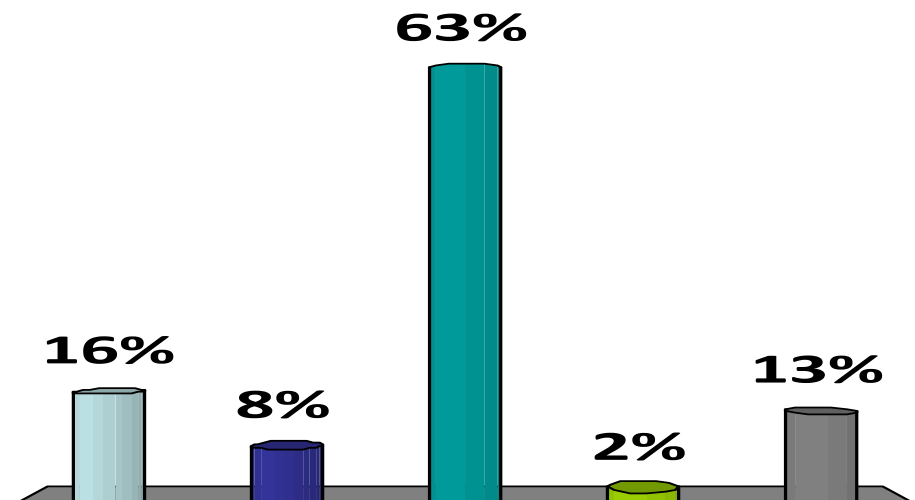
Which of the following is **NOT** a fundamental OOP principle?

1. Inheritance
- ✓ 2. Behavioralism
3. Encapsulation
4. Polymorphism



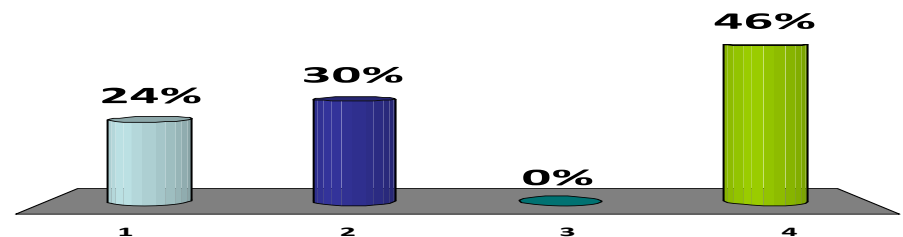
Encapsulation suggests that you should:

- A. Keep each class in its own file.
- B. Keep all of a class's behaviors private.
- ✓ C. Keep all of a class's data private.
- D. Prefer to use static variables
- E. Write you code in a Japanese capsule hotel.



Which statement is **NOT** true?

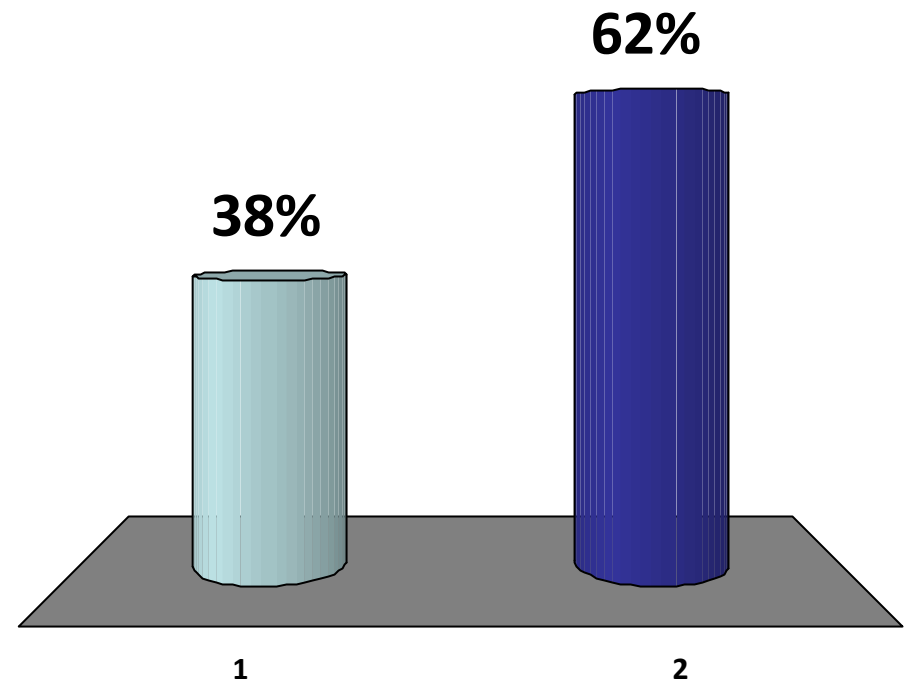
1. All methods in an interface should be public.
2. You may not declare an instance variable in an interface.
3. An interface should be well commented.
- ✓ 4. An interface may include some methods that are completely implemented.



A constructor is the first place a variable is initialized?

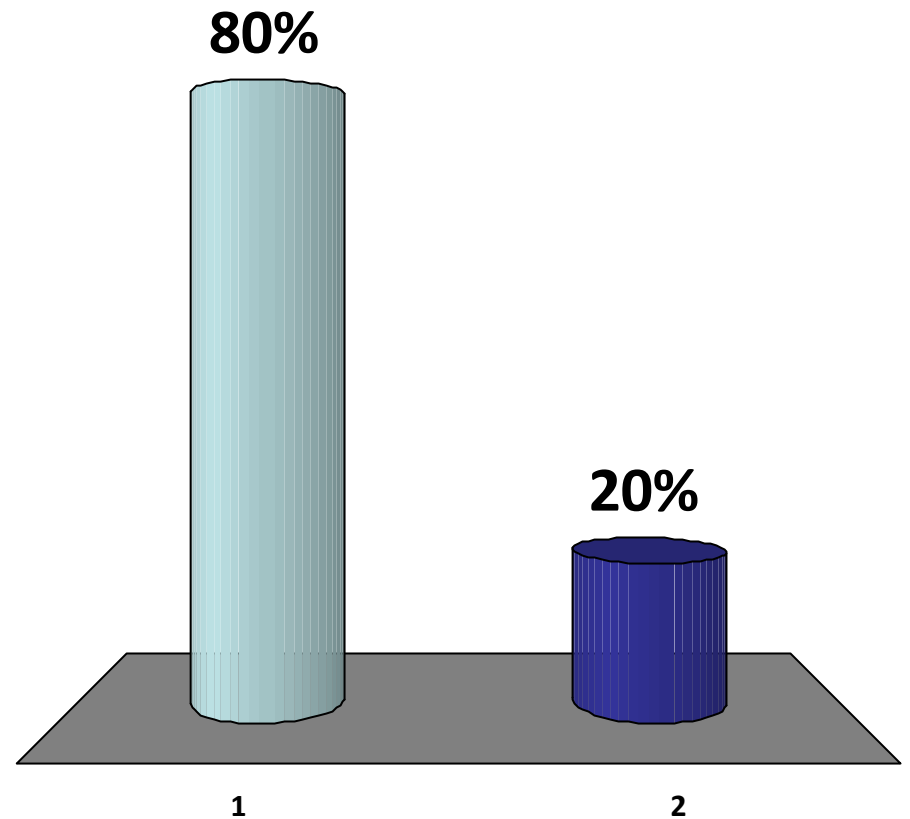
1. Yes

✓ 2. No



You do not need to specify the size of an array when you declare it.

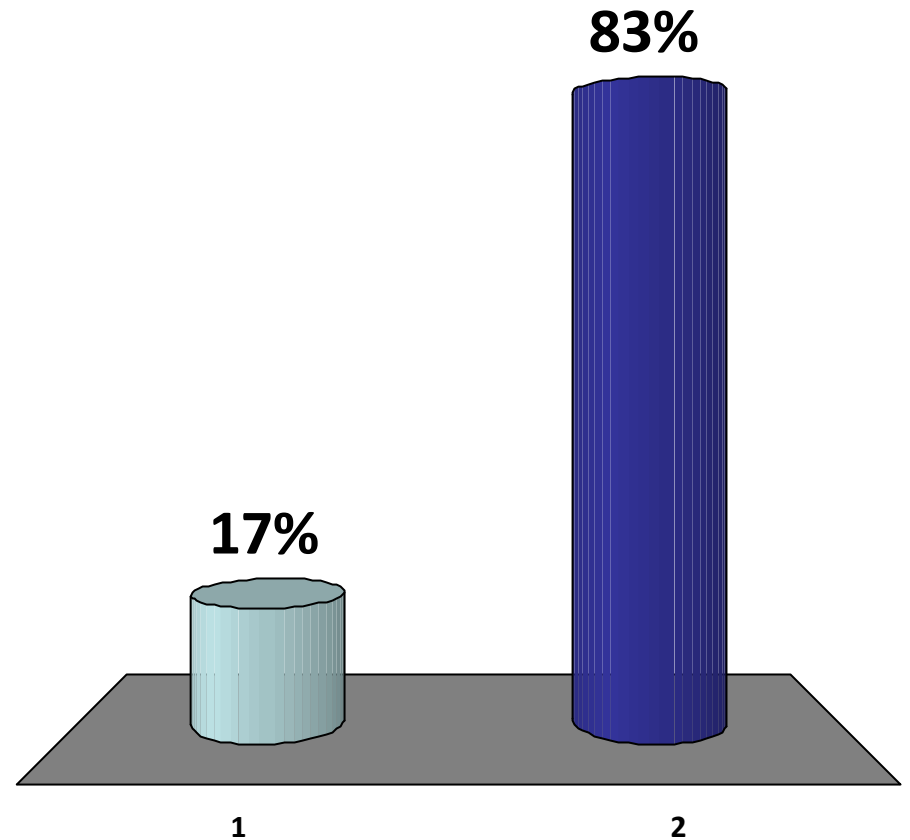
1. True
2. False



A static variable can be accessed **only** by a static method.

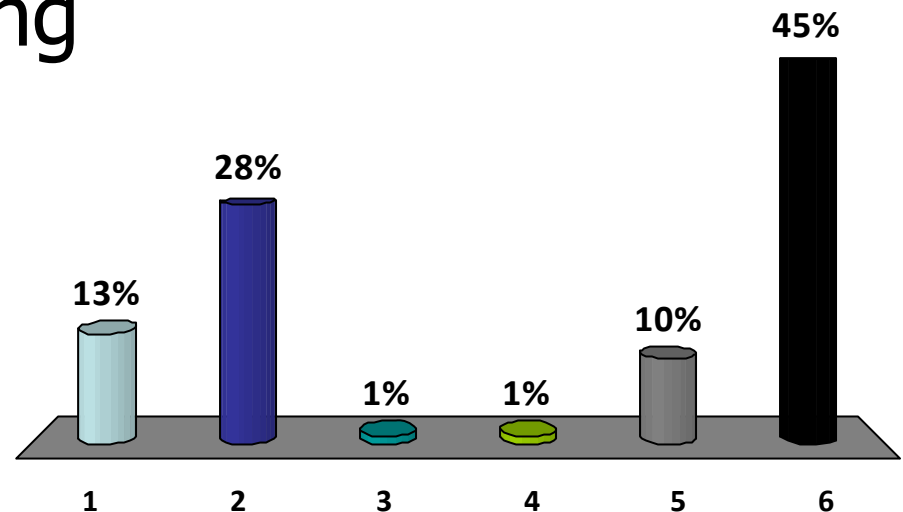
1. True

✓ 2. False



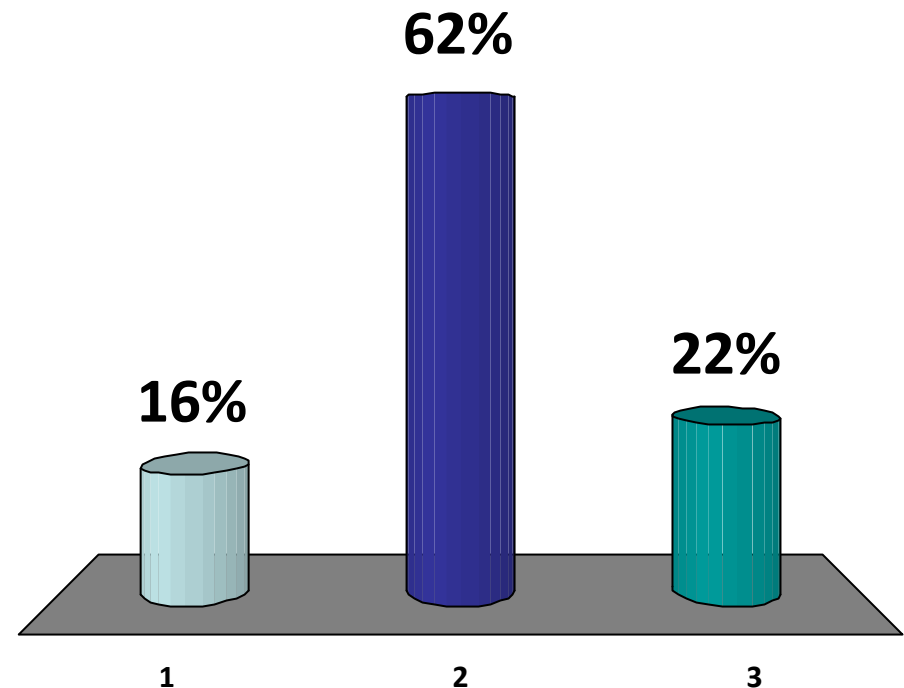
Which of the following is **NOT** a benefit of encapsulation?

1. Modularity
2. Bug reduction
3. Information (data) hiding
4. Implementation hiding
5. Logical code organization
6. None of the above.



Interfaces are primarily useful as a conceptual method of separation, and have little practical impact on my program.

1. True
- ✓ 2. False
3. Sort of



```
class VectorTextFile
```


VectorTextFile

```

/*****
Class: VectorTextFile
Purpose: Represents a text file as a vector, i.e., as a sorted array of
word/count pairs that appear in the text file.

Constructor: VectorTextFile(String fileName)
Behavior: Reads the specified text file and parses it appropriately.

Public Class Methods:
    int Norm() : Returns the norm of the vector.

Static Methods:
    double DotProduct(VectorTextFile A, VectorTextFile B) :
        Returns the dot product of two vectors.
    double Angle(VectorTextFile A, VectorTextFile B) :
        Returns the angle between two vectors.
*****/

// This class is part of the cs2020 package.
package sg.edu.nus.cs2020;

// This class uses the following two packages (associated with reading files):
import java.io.FileInputStream;
import java.io.IOException;

```

Compare Two Documents

Given: documents A and B

1. Read and parse text
2. Create vectors v_A and v_B
3. Calculate norm: $|v_A|$
4. Calculate norm: $|v_B|$
5. Calculate dot product: $(v_A \cdot v_B)$
6. Calculate angle $\Phi(v_A, v_B)$

Document Distance

Basic object/class:

VectorTextFile

Functionality:

- Initialization: Reads in file
- public method: Norm
- static: Dot-product of two vectors
- static: Angle between two vectors

Document Distance

Basic object/class:

VectorTextFile

Public functionality:

```
double norm()
```

```
int DotProduct(VectorTextFile A, VectorTextFile B)
```

```
double Angle(VectorTextFile A, VectorTextFile B)
```

All other functionality is private / internal.

Document Distance Main

```
package sg.edu.nus.cs2020;

public class DocumentDistanceMain
{
    public static void main(String[] args)
    {
        VectorTextFile A = new VectorTextFile("FileOne.txt");
        VectorTextFile B = new VectorTextFile("FileTwo.txt");

        double theta = VectorTextFile.Angle(A,B);

        System.out.println("The angle between A and B is: " + theta + "\n");
    }
}
```


Document Distance

Basic object/class:

VectorTextFile

Constructor: parameter filename

- Reads in file
- Parses file into words
- Sorts words
- Counts word frequencies

Member Variable Declaration

```
// Class declaration:
public class VectorTextFile {

    /**
     * Class member variables
     */

    // Array of words in the file
    String[] m_WordList;

    // Number of words in the file
    int m_FileWordCount;

    // Array of word/count pairs
    WordCountPair[] m_CountedWords;

    // Number of word/count pairs
    int m_WordPairCount;

    // Has the word list been sorted?
    boolean m_Sorted;
```

Constructor

```
//-----  
// Constructor: Reads and parses the specified file  
//  
// Input: String containing a filename  
// Assumptions: fileName is a text file that exists on disk.  
// Properties: On completion, m_WordList contains a sorted array of all the  
// words in the text file, m_FileWordCount is the number of words in the  
// text file, m_CountedWords contains a sorted array of word/count pairs  
// with one entry for every distinct word in the text file, m_WordPairCount  
// is the number of word/count pairs, and the flag m_Sorted is true.  
// Characters in the file are treated in the following manner:  
// (a) Every letter is made lower-case.  
// (b) All punctuation is removed.  
// (c) Each end-of-line marker ('\n') is replaced with a space.  
// (d) All (other) non-letters and non-spaces are removed.  
//-----  
public VectorTextFile(String fileName)  
{  
    // Begin a block of code that handles exceptions (i.e., errors)  
    try{  
  
        // First, initialize class variables  
        m_WordList = null;  
        m_CountedWords = null;  
        m_FileWordCount = 0;  
        m_WordPairCount = 0;  
        m_Sorted = false;  
    }
```


Constructor

```
// Next, read in the file and parse it into words.
ParseFile(fileName);

// Check for errors:
if ((m_FileWordCount < 1) || (m_WordList == null))
{
    throw new Exception("Reading the file failed.");
}

// Next, sort the words.
InsertionSortWords();

// Check for errors:
if (m_Sorted == false)
{
    throw new Exception("Sorting failed.");
}
VerifySort();

// Finally, count the number of times each word appears in the file.
CountWordFrequencies();

// Check for errors:
if ((m_WordPairCount < 1) || (m_CountedWords == null))
{
    throw new Exception("Counting the word frequencies failed.");
}
}
// Catch any exceptions (i.e., errors) and report problems.
catch(Exception e)
{
    System.out.println("Error creating VectorTextFile.");
}
}
```

Document Distance

Secondary object/class: WordCountPair

Encapsulates:

String word

int count

Functionality:

Constructor: sets word and counts

String getWord()

int getCount()

Puzzle of the Week

Imagine three dice:

- A has six sides: 2, 2, 4, 4, 9, 9
- B has six sides: 1, 1, 6, 6, 8, 8
- C has six sides: 3, 3, 5, 5, 7, 7



Game:

- Alice chooses a die.
- Bob chooses a die.
- Alice and Bob both roll.
- The higher value wins.

Questions:

- Which die should Alice choose?
- Which die should Bob choose?
- Who is more likely to win?

Puzzle of the Week

Imagine three dice:

- A has six sides: 2, 2, 4, 4, 9, 9
- B has six sides: 1, 1, 6, 6, 8, 8
- C has six sides: 3, 3, 5, 5, 7, 7



Non-transitive dice!?!?

- A beats B
- B beats C
- C beats A

Puzzle of the Week

Imagine three dice:

- A has six sides: 2, 2, 4, 4, 9, 9
- B has six sides: 1, 1, 6, 6, 8, 8
- C has six sides: 3, 3, 5, 5, 7, 7



Challenge:

- Assume a die has 6 sides.
- Design new dice that maximize the probability that Bob wins.

Puzzle of the Week

Imagine three dice:

- A has six sides: 2, 2, 4, 4, 9, 9
- B has six sides: 1, 1, 6, 6, 8, 8
- C has six sides: 3, 3, 5, 5, 7, 7



Challenge:

- Assume a die has 6 sides.
- What is the largest set of dice x_1, x_2, \dots, x_n such that $x_1 > x_2 > \dots > x_n > x_1$

For next time...

Problem Session:

- Today!

Wednesday:

- More OOP
- Inheritance
- Lists

Problem Set 1:

- Released. Due next Tuesday night.

Public Methods

```
/* *****  
 * Public Class Methods      *  
***** */  
  
//-----  
// Norm: Returns the norm of the vector.  
//  
// Input: None.  
// Output: The norm of the vector.  
// Assumptions: m_CountedWords contains a sorted list of distinct  
// word/count pairs, and m_WordPairCount contains the number of word/count  
// pairs.  
// Methodology: The norm of a vector X is defined to be the square-root of  
// DotProduct(X,X) .  
//-----  
public double Norm()  
{  
    int dot = VectorTextFile.DotProduct(this, this);  
    return Math.sqrt(dot);  
}
```


Checking for errors

```
private void VerifySort() throws Exception
{
    // First, check if the sort flag is correctly set:
    if (m_Sorted == false)
    {
        throw new Exception("VerifySort fails: list not sorted.");
    }
    // Next, check if there are any words to be sorted:
    if ((m_FileWordCount < 1) || (m_WordList == null))
    {
        throw new Exception("VerifySort fails: list does not contain any words.");
    }

    // Finally, iterate through the list of words and make sure that they
    // are in properly sorted order.
    for (int i=0; i<m_FileWordCount-1; i++)
    {
        if (m_WordList[i].compareTo(m_WordList[i+1]) > 0)
        {
            throw new Exception("VerifySort fails: list badly sorted.");
        }
    }
}
```


String

Creating strings:

```
String str = "Some text.";
```

```
String altStr = new String("some text");
```

Accessing a string:

- `charAt(int index)`
- `substring(int begin, int end)`
- `toCharArray()`
- `length()`

String

Comparing strings:

- `compareTo(String otherString)`
- `compareToIgnoreCase(String otherString)`
- `equals(Object anObject)`

Using strings:

- Flexible and easy: `str = str + 'c';`
- Use with care...