

Today only!

**Learn to program in Java in
(less than) one hour!**

Special deal!!

Today only!

**Learn to program in Java in
(less than) one hour!**

3 EASY PAYMENTS

**VERY INTENSE DROP SHADOW
AND EVERYWORD IS CAPITALIZED!**



1-800-BUY-THIS

u/briekingher

CS2040S

Data Structures and Algorithms

Welcome!

Reminders

1. Tutorials / Recitations were announced last night.

- Timeslot was fixed.
- Section may change. (Balancing will finish today.)

2. DO NOT use ModReg to adjust or swap tutorial or recitation.

- Not your allocated slot. Not any slot.
- It will only cause trouble!

Reminders

3. Help! I don't have a slot!

- Please fill out the appeal on Coursemology.
- Deadline: Thursday (tomorrow), noon.

4. Help! My assigned slot conflicts!

- Please fill out the appeal on Coursemology.
- Deadline: Thursday (tomorrow), noon.

Reminders

5. Help! I don't like my slot!

- Please find someone else to swap with.
- See Coursemology forum.
- Please fill out the appeal on Coursemology.
- Deadline: Thursday (tomorrow), noon.

Reminders

5. Help! I don't like my slot!

- Please find someone else to swap with.
- See Coursemology forum.
- Please fill out the appeal on Coursemology.
- Deadline: Thursday (tomorrow), noon.

Eldric will be handling appeals through Coursemology. Expect to have the results later on the same Thursday, some time after 4PM.

Reminders

DO NOT use ModReg to adjust or swap tutorial or recitation.

- We have the master list.
- We will revert back changes you make directly.
- The only way to change is via Coursemology appeal survey.

Reminders

Do read Coursemology Announcements.

- We will keep you updated on the status.

Reminders

Problem Set 1:

- Available on Coursemology
- Full description in pdf file (with FAQ at end).
- Due Sunday night.
- Discussion on Coursemology if you have questions
 - Java setup issues (JDK, etc.)
 - Why is my image blank?
 - Why is my image wrong?
 - I don't recognize the person in that picture?

Reminders

Beware:

Finalize Submission means...

You have finalized your submission.

- No more changes after it is finalized.
- Only click finalize when you are done with all the parts of the problem set or lecture review.

On CS2040S Problem Sets:

Do not use libraries that simplify the problem.

Do not use libraries unless the problem set specifically says you can.

In the future, you can ask on the forum and/or check with your tutor, if unsure.

If the goal of the problem set is to write a sorting routine, then calling the Java library sort defeats the purpose...

Announcements

- **Update:** Tutorials will start from Week 3 (not from Week 4)
- In-person tutorials will start on Week 4. We will post a video recording explaining Week 3 tutorial.

Today only!

**Learn to program in Java in
(less than) one hour!**

3 EASY PAYMENTS

**VERY INTENSE DROP SHADOW
AND EVERYWORD IS CAPITALIZED!**



1-800-BUY-THIS

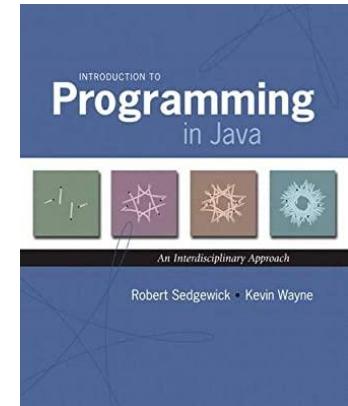
u/briekingher

the basics of OOP and Java

the basics of OOP and Java

For more advanced topics:

- See CS2030S.
- Google.
- See suggested (optional) textbook.



Java “advanced” features:

Examples:

Lambda expressions:

```
void sortSomeThings(int[] array) {  
  
    MyComparator compareFunction = (a, b) -> {  
        if (a < b) return true;  
        else return false;  
    };  
  
}
```

Java “advanced” features:

Examples:

Type inference:

```
var complicatedMap = new HashMap<String, List<String>>();
```

VS.

```
HashMap<String, List<String>> complicatedMap = new HashMap<String, List<String>>();
```

Java “advanced” features:

Examples:

Default, static private methods in an interface:

Many different use cases, restrictions, rules, best practices, ...

Advice:

Do not use the advanced features:

- Typically, just makes code shorter.
- Very little extra functionality.
- Often hide what is really happening.
- Can make code easier to read, but can make code harder to read.

* Especially if you are new to Java.

Goals in writing code:

1. Correct / bug-free.
2. Easy to read / understand.
3. Efficient.
4. Submitted by the deadline.
5. ..
6. ..
7. ..
8. ..
100. Short.

Advice:

Make your code *intentional*.

(Do not rely on default / non-explicit behavior.)

* Especially if you are new to Java.

More advice: use the IntelliJ debugger

The screenshot shows the IntelliJ IDEA interface with the "GradientTest.java" file open. A red breakpoint is set on line 22. A yellow arrow points from the text "Set a breakpoint..." to the red dot on line 22. Another yellow arrow points from the text "Click 'debug'" to the green gear icon in the toolbar, which is highlighted with a blue box.

```
10     java.util.Random rGen = new Random();
11
12     float[][] points;
13     boolean[] classify;
14
15     // Equation for the line: x/xInt + y/yInt = 1
16     GenerateData(float x, float y){
17         xInt = x;
18         yInt = y;
19     }
20
21     boolean testPoint(float x, float y){
22         float a = (x/xInt);
23         float b = (y/yInt);
24         if ((a+b) >= 1) return true;
25         else return false;
26     }
27
28     void generate(int count){
29         points = new float[count][2];
30         classify = new boolean[count];
31         float range = (maxX - minX);
32         float mid = range/2;
```

Set a breakpoint...

Click “debug”

More advice: use the IntelliJ debugger

The screenshot shows the IntelliJ IDEA debugger interface during a step-through session. The code editor displays a Java file with the following content:

```
19 }  
20  
21     boolean testPoint(float x, float y){ x: -38.506508 y: 56.380707  
22     float a = (x/xInt); a: -7.7013016 x: -38.506508 xInt: 5.0  
23     float b = (y/yInt); b: 11.276141 y: 56.380707 yInt: 5.0  
24     if ((a+b) >= 1) return true; a: -7.7013016 b: 11.276141  
25     else return false;  
26 }  
27  
28     void generate(int count){  
29         points = new float[count][2];  
30         classify = new boolean[count];  
31         float range = (maxX - minX);  
32         float mid = range/2;  
33  
34         for (int i=0; i<count; i++){  
35             points[i][0] = rGen.nextFloat()*range - mid;  
GenerateData > testPoint()
```

The line `if ((a+b) >= 1) return true;` is highlighted in blue, indicating it is the next line to be executed. A red breakpoint icon is visible on line 22.

The bottom panel shows the Variables tool window with the following list of variables:

Variables
this = {GenerateData@621}
x = -38.506508
y = 56.380707
a = -7.7013016
b = 11.276141
yInt = 5.0

A callout box with the text "Step through your code" is positioned over the Variables window.

Warning:

Today's goal:

To make you aware of key aspects of Java

After class:

Go look things up!

the basics of OOP and Java

Programming Paradigms

Programming paradigms:

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

How to organize information?

How to think about a solution?

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

Abstraction

Separate interface ("what it is supposed to do") from implementation ("how it does it").

Encapsulation

Hide implementation. Only make interface publicly visible.

Inheritance

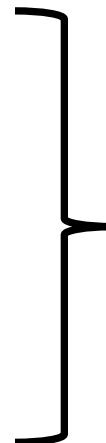
Build new classes by extending existing classes. (Share functionality.)

Polymorphism

Same interface, but different behavior based on context.
`(animal.vocalize())` meows if animal is a cat, and barks if animal is a dog..)

Object-oriented Paradigm

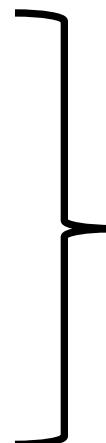
Abstraction



Very relevant
to CS2040S

Encapsulation

Inheritance



Less relevant
to CS2040S
(but very important overall).

Polymorphism

Naruto the new hire

Description of Naruto:

- “Nice guy!”
- “Really likes bananas!”
- “Not the smartest fellow... but friendly!”
- “I’m afraid he’s going to \$%^& up our code, man!”



What should we do?

Give him some pointless work!

Send him back to the forest!

But we want Naruto to help with
our project!

But we don't want him to wreck
our software.



Abstraction

Remove all unnecessary elements:

- What Naruto needs-to-know? **Expose!**
- What Naruto doesn't need-to-know? **Hide!**

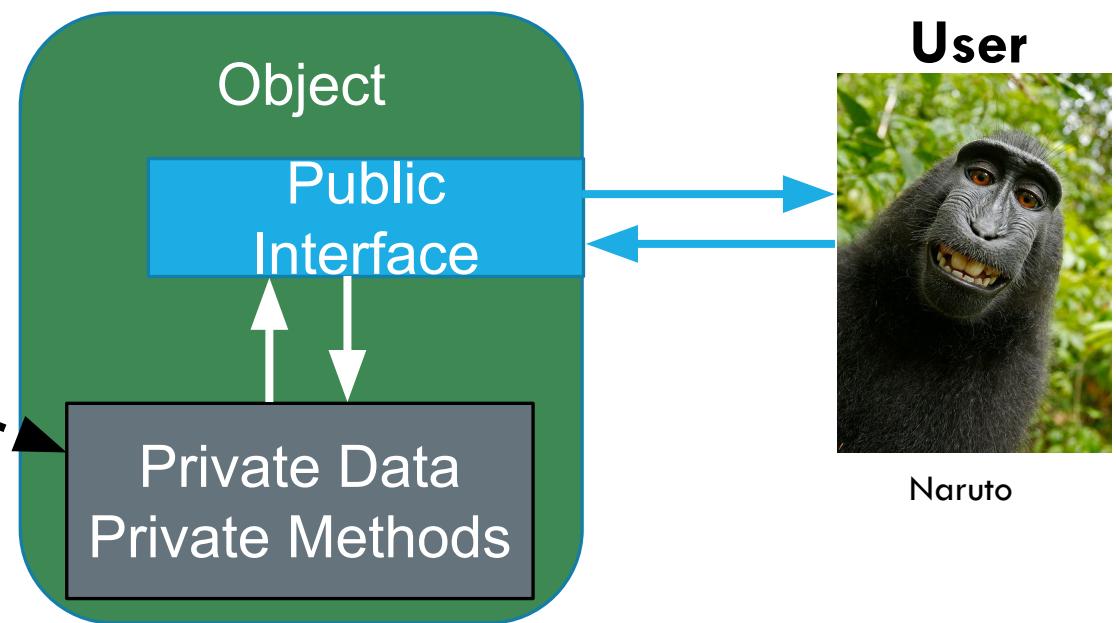
Keep things simple!

Encapsulation and information hiding

Implementer



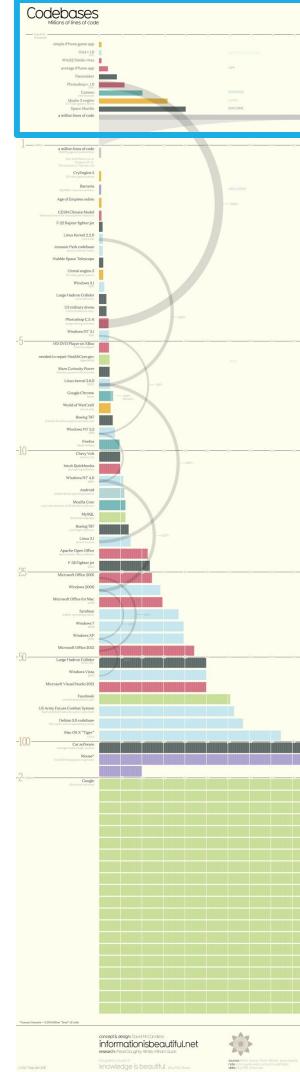
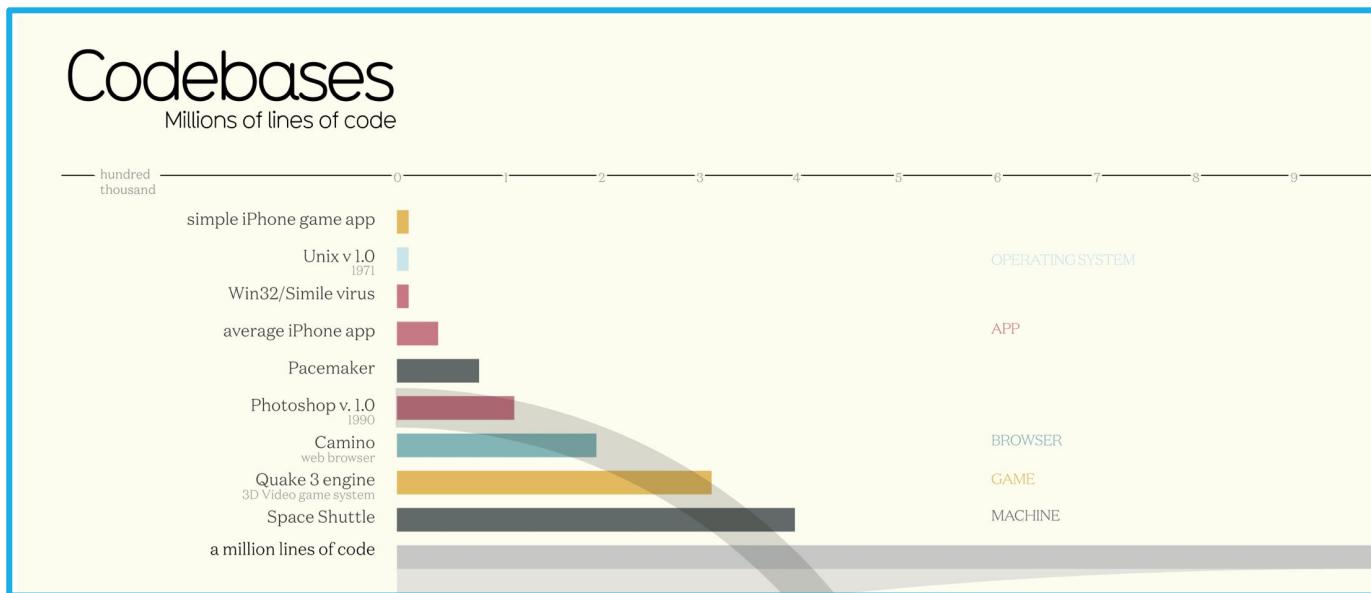
[XKCD: Black Hat]



Claim: We are all Naruto!

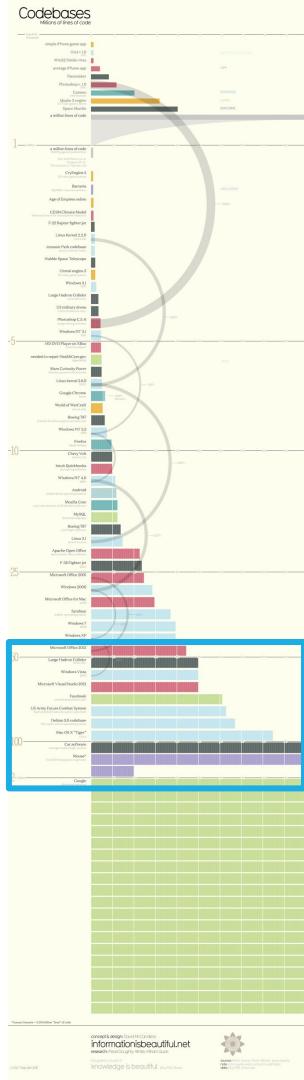
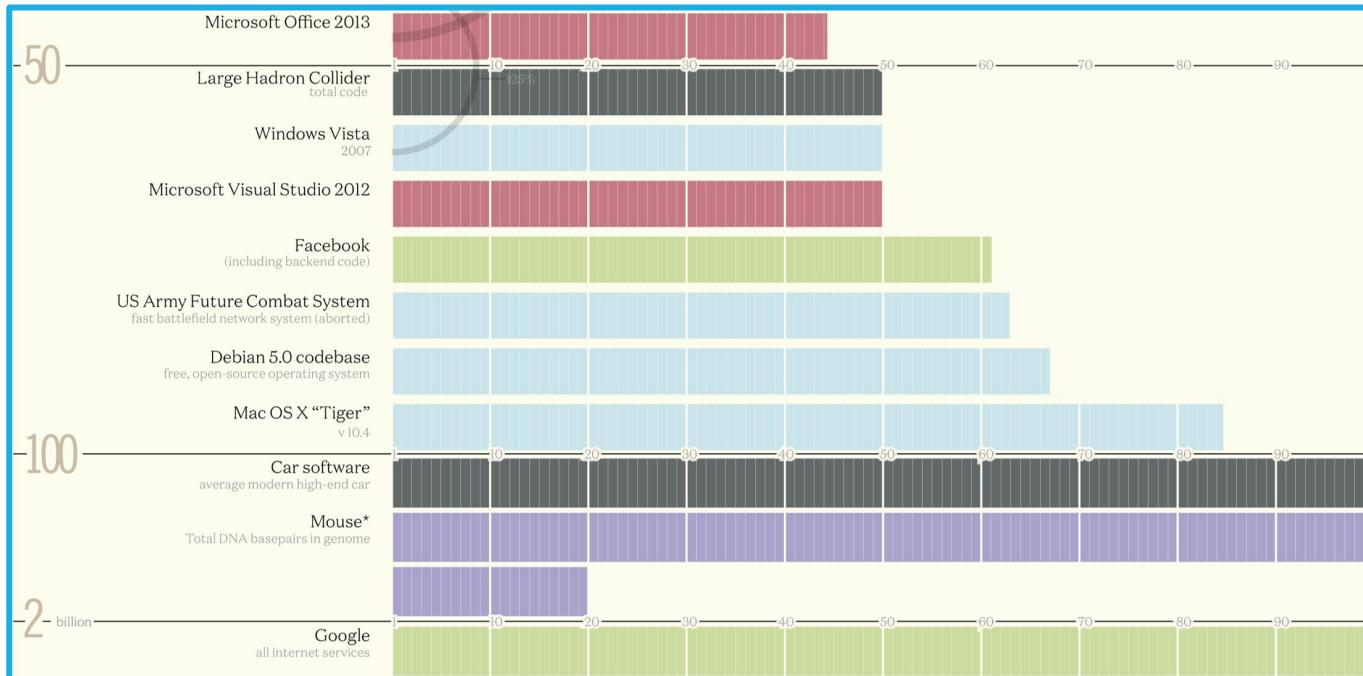


Software is getting very complex



[<http://www.informationisbeautiful.net/visualizations/million-lines-of-code/>]

Software is getting very complex



[<http://www.informationisbeautiful.net/visualizations/million-lines-of-code/>]

Software is getting very complex

How many lines of code in Google?

The **Google** codebase includes approximately one billion files and has a history of approximately 35 million commits spanning **Google's** entire 18-year existence. The repository contains 86TB^a of data, including approximately two billion **lines of code** in nine million unique source files. Jun 28, 2016

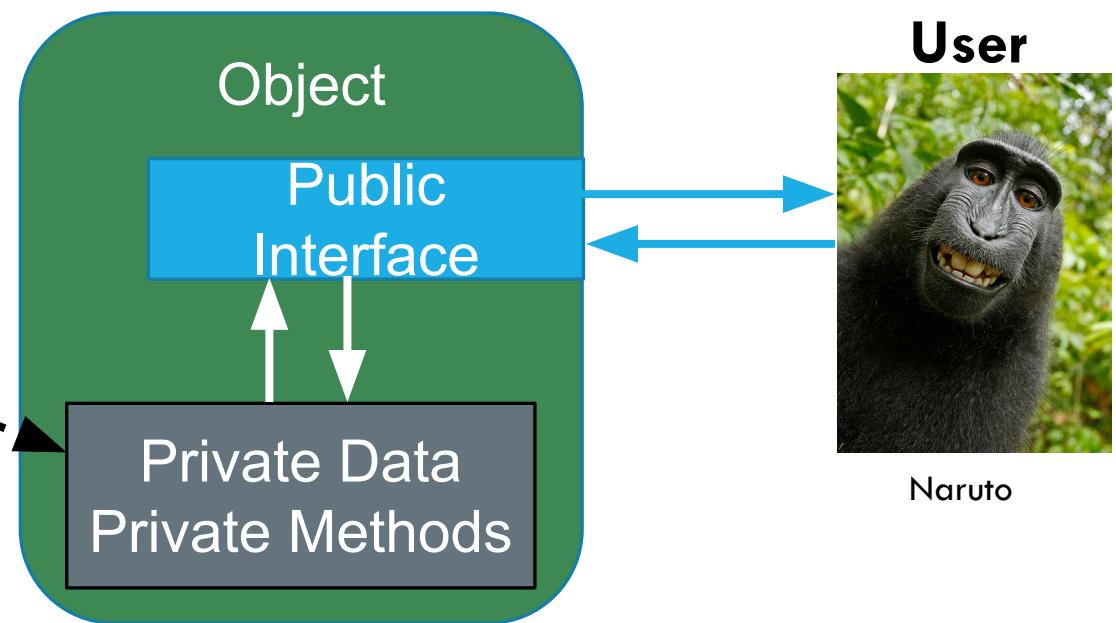
Also look at : <http://www.informationisbeautiful.net/visualizations/million-lines-of-code/>

Encapsulation and information hiding

Implementer



[XKCD: Black Hat]



Abstract away unnecessary details

Better understand complex software

Save us from ourselves

Abstraction

Software engineering

- Divide problem into components.
 - Define *interface* between components.
 - Assign a team to build each component.
 - (Recurse.)
-
- Top down design: get the big idea first, then figure out how to implement it.

Abstraction

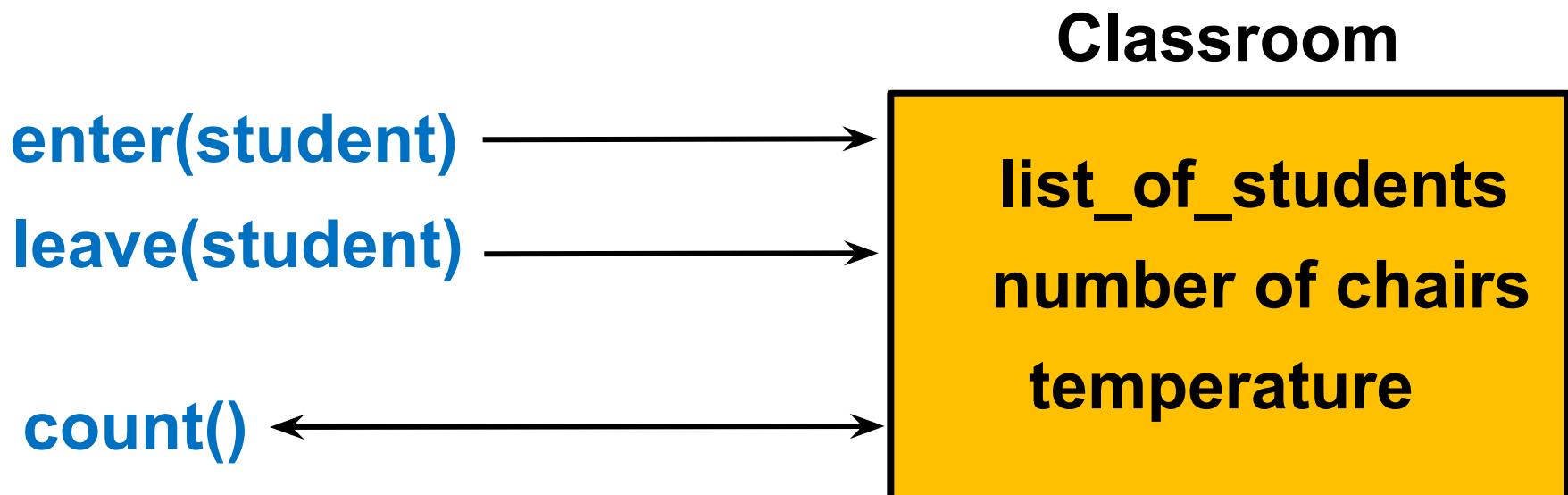
Algorithm design

- Divide problem into components.
- Define *interface* between components.
- Solve each problem separately.
- (Recurse.)
- Combine solutions.

Object-oriented Programming

Object has:

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



How to implement a **File System**?

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

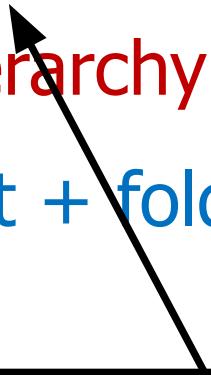
How to implement a **File System**?

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

Objects represent state (“nouns”) not actions (“verbs”).

How to implement a **File System**?

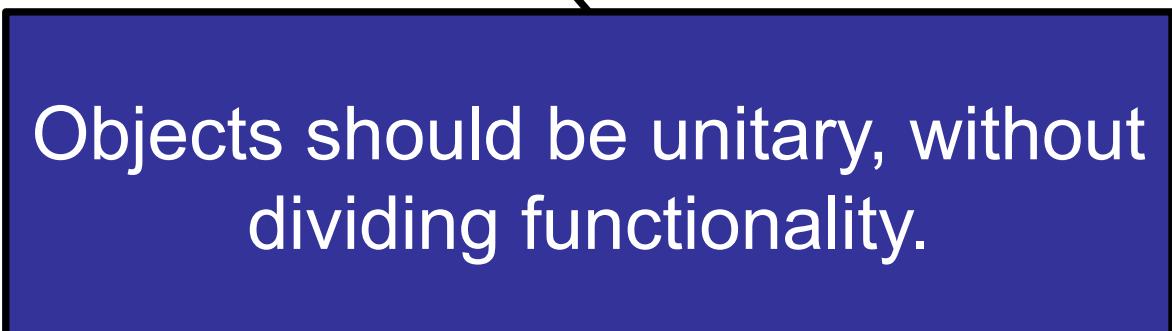
1. ~~file management object + file contents object~~
2. ~~file object~~
3. ~~folder hierarchy + folder contents~~
4. ~~file object + folder object~~



What about folders?

How to implement a **File System**?

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



Objects should be unitary, without dividing functionality.

How to implement a **File System**?

1. ~~file management object + file contents object~~
2. ~~file object~~
3. ~~folder hierarchy + folder contents~~
4. file object + folder 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 »

First principle of Java

« Everything is an object »

*“But I was told Java 8+ was functional!”
“What about the lambda expressions?”
“Java supports anonymous functions now!”*



To Java, these are just disguised objects.

First principle of Java

« Everything is an object »

A class is a template for producing an object.

Defining a class in Java

```
class File
{
    String name = "";
    FileData contents = null;

    void rename(String newName) { ... }

    FileData getData() { ... }

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

Defining a class in Java

```
class File
{
    String name = "";
    FileData contents = null;

    void rename(String newName) { ... }

    FileData getData() { ... }

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

Not Java syntax.
Abbrv. for slides.

Defining a class in Java

```
class File
```

```
{
```

```
    String name = "";
```

```
    FileData contents = null;
```

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

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

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

```
}
```

Defining a class in Java

```
class File  
{  
    String name = "";  
    FileData content;  
  
    void rename(String newName) { ... }  
  
    FileData getData() { ... }  
  
    void setData(FileData newdata) { ... }  
}
```

Name of class **MUST**
be name of file.



File.java

Defining a class in Java

```
class File  
{
```

```
    String name = "";  
    FileData contents = null;
```

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

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

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

```
}
```

Variables initialized when defined first.

Advice: don't rely on defaults: be explicit.

Second principle of Java

« Everything has a type »

Second principle of Java

« Everything has a type »

“But I can declare a local variable with just var as an unspecified type!”



To Java, there is still a type. It just sometimes guesses it for you.

Second principle of Java

« Everything has a type »

```
int j = 7;  
j = "7";
```

ERROR

```
var j = 7;  
j = "7";
```

Second principle of Java

« Everything has a type »

```
int j = 7;  
j = "7";
```

ERROR

```
var j = 7;  
j = "7";
```

Advice: always specify the type.
(Don't rely on type inference.)

Types

```
class File
{
    String name = "";
    FileData contents = null;

    void rename(String newName) { ... }

    FileData getData() { ... }

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

Java library class: 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()`

[https://docs.oracle.com/en/java/javase/21/docs
/api/java.base/java/lang/String.html](https://docs.oracle.com/en/java/javase/21/docs/api/java.base/java/lang/String.html)

Java library class: String

Comparing strings:

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

Using strings:

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

[https://docs.oracle.com/en/java/javase/21/docs
/api/java.base/java/lang/String.html](https://docs.oracle.com/en/java/javase/21/docs/api/java.base/java/lang/String.html)

Java library class: String

Comparing strings:

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

Using strings

- Flexible
- Use with

Common bug:

```
String myName = "Seth";
String profName = "Seth";

if (myName == profName)
{
    System.out.println("This will not always work correctly.");
}
```

Types

```
class File
{
    String name = "";
    FileData contents = null;

    void rename(String newName) { ... }

    FileData getData() { ... }

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

Member (Instance) Variables

```
class File
{
    String name = "";
    FileData contents = null;

    void rename(String newName) { ... }

    FileData getData() { ... }

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

Methods (Functions)

```
class File
{
    String name = "";
    FileData contents = null;

    void rename(String newName){...}

    FileData getData( ){...}

    void setData(FileData newdata){...}

}
```

Defining a class in Java

```
class File
{
    String name = "";
    FileData contents = null;

    void rename(String newName) { ... }

    FileData getData() { ... }

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

Another class

```
class Folder
{
    String name;
    Folder[] children;
    File[] files;
    intgetNumFiles() { ... }
    File getFile(int i) { ... }
}
```

Note array notation

Note “recursive” folder

Class vs. Object

What's the difference?

Class vs. Object

What's the difference?

Class is a *template* for how to make an object.

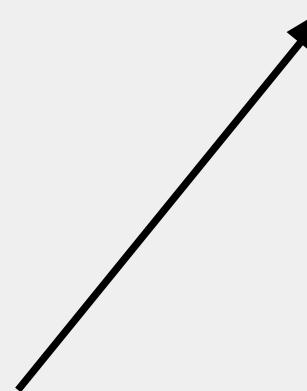
An object is an instance of the class.

Creating a new object

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

Creating a new object

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



Calls “constructor” with parameter: *name*

Constructors

```
class File
{
    String name = "";
    FileData contents = null;

    File(String fileName)
    {
        name = fileName;
        contents = null;
    }
}
```

Constructor:

- Same name as class.
- Takes 0 or more parameters.
- Called on object creation.
- Used to initialize class.
- Runs after variables initialized on declaration.

Constructors

```
class File
{
    String name = "FileData content";
    File(String fileName)
    {
        name = fileName;
        contents = null;
    }
}
```

Many rules involving constructors.

E.g., when exactly are they executed during object construction? In what order? Etc.

Especially complicated with inheritance.

Object-oriented Paradigm

Abstraction

Separate interface ("what it is supposed to do") from implementation ("how it does it").

Encapsulation

Hide implementation. Only make interface publically visible.

Inheritance

Build new classes by extending existing classes. (Share functionality.)

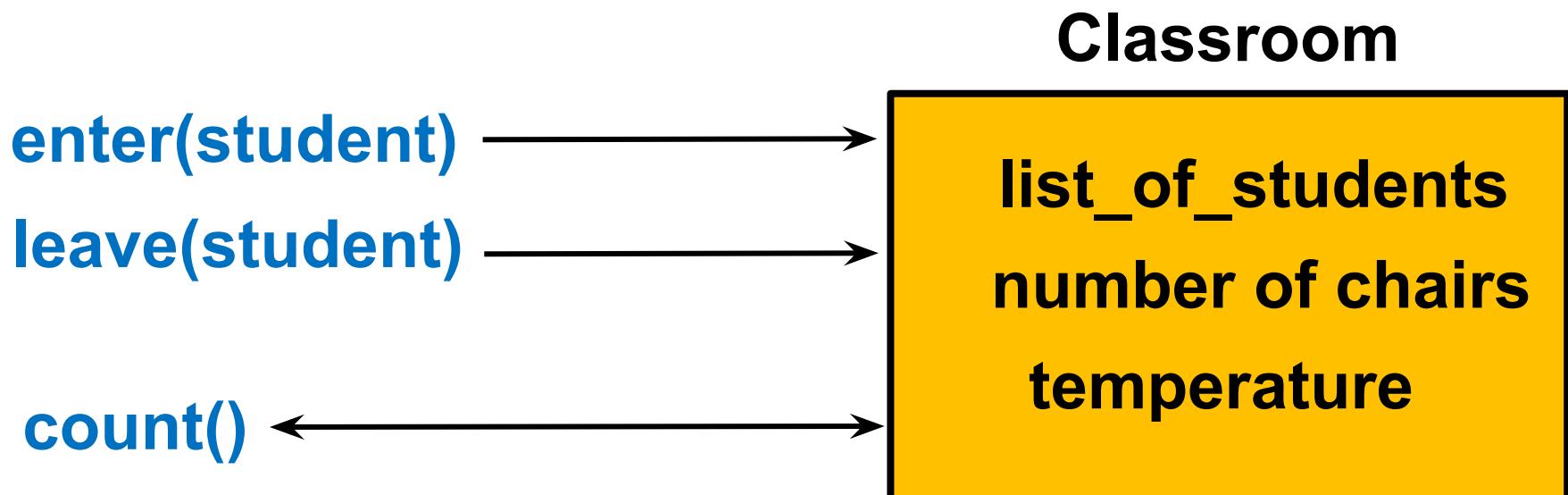
Polymorphism

Same interface, but different behavior based on context.
`(animal.vocalize())` meows if animal is a cat, and barks if animal is a dog..)

Object-oriented Programming

Object has:

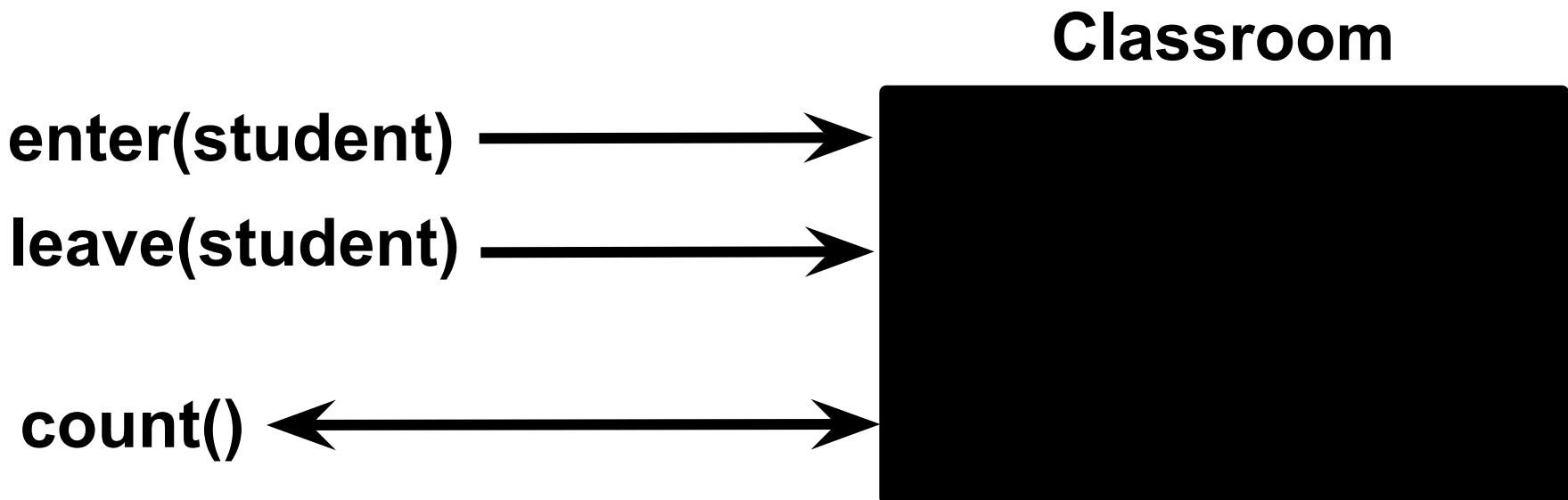
- State (i.e., data)
- Behavior (i.e., methods for modifying the state)



Abstraction

Interface: how you manipulate the object

Implementation: details hidden inside the object



Defining an interface

```
// Explain with a comment  
// what your interface is for.  
interface IFile  
{  
    // Comments explain how to use interface  
    void rename(String newName);  
  
    FileData getData();  
  
    void setData(FileData newdata);  
}
```

*Note no functionality!
Only method names.*



*Except for the
“advanced features.”*

Implementing an interface

```
class File implements IFile
{
    String name = "";
    FileData contents = null;

    void rename(String newName) { ... }

    FileData getData() { ... }

    void setData(FileData newdata)
}
```

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

```
interface IFile
{
    void rename(String newName);

    FileData getData();

    void setData(FileData newdata);
}
```

Implementing an interface

```
class OtherFile implements IFile  
{  
    char[] nom;  
  
    char[] meteo;  
  
    FileData getData() { . . . }  
  
    void setData(FileData nouveau) { . . . }  
}
```

```
interface IFile  
{  
    void rename(String newName);  
  
    FileData getData();  
  
    void setData(FileData newdata);  
}
```

Implementing an interface

```
class OtherFile implements IFile  
{
```

```
    char[] nom;
```

Is this good?

```
    char[] meteo;
```

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

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

```
}
```

```
interface IFile  
{  
    void rename(String newName);  
    FileData getData();  
    void setData(FileData newdata);  
}
```

Implementing an interface

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

    char[] meteo;

    FileData getData() { . . . }

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

Error!

```
interface IFile
{
    void rename(String newName);

    FileData getData();

    void setData(FileData newdata);
}
```

Implementing an interface

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

    char[] meteo;

    void rename() { ... }

    FileData getData() { ... }

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

```
interface IFile
{
    void rename(String newName);

    FileData getData();

    void setData(FileData newdata);
}
```

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.

Problem Set 1

```
package cs2040;

/*
 *
 * @author gilbert
 *  Interface: ILFShiftRegister
 * Description: a linear feedback shift register based on XOR with one tap
 *
 */
public interface ILFShiftRegister {

    // Sets the value of the shift register to the specified seed.
    public void setSeed(int[] seed);

    // Shifts the register one time, returning the low-order bit.
    public int shift();

    // Shifts the register k times, returning a k-bit integer.
    public int generate(int k);

}
```

```
///////////
// This is the main shift register class.
// Notice that it implements the ILFShiftRegister interface.
// You will need to fill in the functionality.
///////////

/*
 * class ShiftRegister
 * @author
 * Description: implements the ILFShiftRegister interface.
 */

public class ShiftRegister implements ILFShiftRegister {
    ///////////
    // Create your class variables here
    ///////////
    // TODO:

    ///////////
    // Create your constructor here:
    ///////////
    ShiftRegister(int size, int tap) {
        // TODO:
    }

    ///////////
    // Create your class methods here:
    ///////////
    /**
     * setSeed
     * @param seed
     * Description:
     */
    @Override
    public void setSeed(int[] seed) {
        // TODO:
    }
```

Quick summary...

So far: Object-Oriented Programming

- Defining classes and interfaces
- Implementing interfaces
- Using interfaces

Next: Some Java Details

- 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()
}
```

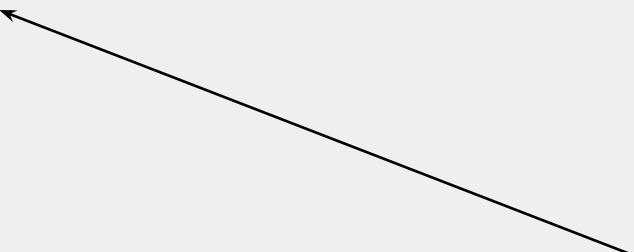
Access Control

- (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 secretFunction();
}

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

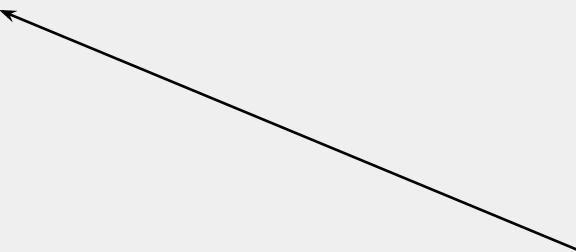


Error: cannot access secretFunction().

Access Control

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

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

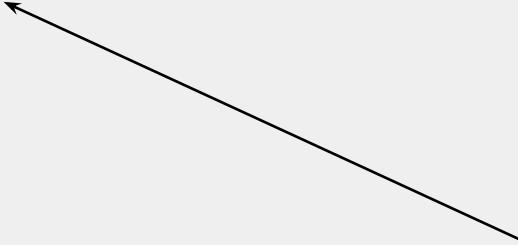


Error: cannot access secretVariable.

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
{
    public int secretVariable;
}
```

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

Bad idea!

Breaks encapsulation.

Ok, can access secretVariable.

Access Control

```
public class B
{
    public B(int data) {
        // Initialize class B using data.
    }

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

A constructor should (almost) always be **public**.

Why?

Access Control

```
public interface ISee  
{  
  
    public int ReadSomething(int data);  
  
    public int WriteSomething(int data);  
}
```

An interface should (almost) always be **public**.
Interface methods should (almost) always be **public**.

Why?

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

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

Advice:

Always specify the access you intend
(even if the default behavior is okay).

Packages

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

For CS2040S:

We will not use packages.

Importing library code

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

Import everything from CleverCode.

Importing library code

```
import CleverCode.ShiftRegister;

public class B
{
    public int stealSecrets(A example) {
        ShiftRegister reg = A.getRegister();
    }
}
```

Import just ShiftRegister from package.

Good practice: only import what you need.

Importing library code

```
import java.util.HashMap;

public class B
{
    public int stealSecrets(A example) {
        ShiftRegister reg = A.getRegister();
    }
}
```

Import HashMap from java libraries.

On problem sets: see instructions to see what you can use.

On CS2040S Problem Sets:

Do not use libraries if they make the problem set easier.

Do not use libraries unless the problem set specifically says you can.

If the goal of the problem set is to write a sorting routine, then calling the Java library sort defeats the purpose...

Class vs. Object

What's the difference?

Class is a *template* for how to make an object.

An object is an instance of the class.

Class vs. Object

What's the difference?

Class is a *template* for how to make an object.

An object is an instance of the class.

regular variables/functions are **PER OBJECT**

static variables/functions are **PER CLASS**

static methods

```
class File
{
    private String fileName = "";
    private FileData contents = null;

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

static methods

```
class File
{
    private String fileName = "";
    private FileData contents = null;

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

static methods

```
class File
{
    private String fileName = "";
    private FileData contents = null;

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

Cannot access member variable.

Error!

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 name = "";
    private FileData contents = null;

    public File(String fileName) {
        name = fileName;
        contents = null;
    }
}
```

Initializing an object

```
class File
{
    private String name = "";
    private FileData contents = null;
    // Constructor
    public File(String fileName) {
        name = fileName;
        contents = null;
    }
}
```

Initializing an object

```
class File
{
    public File(String fileName) {
        name = fileName;
        contents = null;
    }
    public File() {
        name = null;
        contents = null;
    }
}
```

Multiple constructors with different signatures.

Initializing an object with an array

```
class File  
{  
    private int [] 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[] pageNumbers = null;  
  
    public File(int NumPages) {  
        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)

Problem Set 1

```
package cs2040;

/*
 *
 * @author gilbert
 *  Interface: ILFShiftRegister
 * Description: a linear feedback shift register based on XOR with one tap
 *
 */
public interface ILFShiftRegister {

    // Sets the value of the shift register to the specified seed.
    public void setSeed(int[] seed);

    // Shifts the register one time, returning the low-order bit.
    public int shift();

    // Shifts the register k times, returning a k-bit integer.
    public int generate(int k);

}
```

```
///////////
// This is the main shift register class.
// Notice that it implements the ILFShiftRegister interface.
// You will need to fill in the functionality.
///////////

/*
 * class ShiftRegister
 * @author
 * Description: implements the ILFShiftRegister interface.
 */

public class ShiftRegister implements ILFShiftRegister {
    ///////////
    // Create your class variables here
    ///////////
    // TODO:

    ///////////
    // Create your constructor here:
    ///////////
    ShiftRegister(int size, int tap) {
        // TODO:
    }

    ///////////
    // Create your class methods here:
    ///////////
    /**
     * setSeed
     * @param seed
     * Description:
     */
    @Override
    public void setSeed(int[] seed) {
        // TODO:
    }
```

```
/*
 * generate
 * @param k
 * @return
 * Description:
 */
@Override
public int generate(int k) {
    // TODO:
    return 0;
}

/*
 * Returns the integer representation for a binary int array.
 * @param array
 * @return
 */
private int toBinary(int[] array) {
    // TODO:
    return 0;
}
```

A few common problems

Library setup

See forum for discussion of making sure IntelliJ is setup correctly (with access to JUnit library, etc.).

A few common problems

The file does not run properly

You need a “main” method.

Make sure IntelliJ is running the right “main” method.

Make sure the filename is the same as the class.

A few common problems

The file does not compile

Check all the red squiggles. (Hover over them.)

Check all the files in the project.

A few common problems

The image file comes up empty.

Check your ShiftRegister.

Check if the image file is in the right place.

Check if the image is being opened correctly (or if there is an exception being thrown).

A few common problems

Other problems?

See discussion in forum.

Google weird error messages.

Ask questions in forum.

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>

Today only!

**Learn to program in Java in
(less than) one hour!**

3 EASY PAYMENTS

**VERY INTENSE DROP SHADOW
AND EVERYWORD IS CAPITALIZED!**



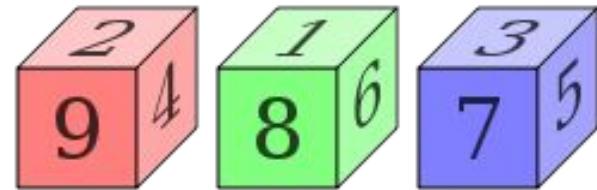
1-800-BUY-THIS

u/briekingher

Puzzle of the Week

Non-transitive Dice: A > B > C > A

- 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.

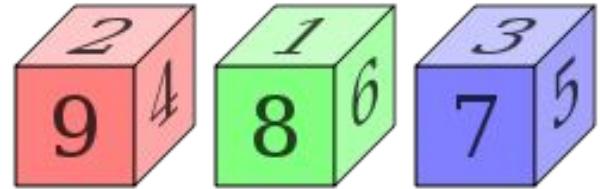
Questions:

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

More Dice Puzzles

Sicherman Dice

- Two six-sided dice
- Same distribution for the SUM



Wikipedia:

Sicherman dice

Article [Talk](#)

From Wikipedia, the free encyclopedia

Sicherman dice /sɪkər'mən/ are a pair of 6-sided dice with non-standard numbers—one with the sides 1, 2, 2, 3, 3, 4 and the other with the sides 1, 3, 4, 5, 6, 8. They are notable as the only pair of 6-sided [dice](#) that are not [normal dice](#), bear only [positive integers](#), and have the same [probability distribution](#) for the [sum](#) as normal dice. They were invented in 1978 by George Sicherman of Buffalo, New York.

Mathematics [\[edit\]](#)

A standard exercise in elementary combinatorics is to calculate the number of ways of rolling any given value with a pair of fair six-sided [dice](#) (by taking the [sum](#) of the two rolls). The table shows the number of such ways of rolling a given value n :

n	2	3	4	5	6	7	8	9	10	11	12
# of ways	1	2	3	4	5	6	5	4	3	2	1

Crazy dice is a [mathematical exercise](#) in elementary [combinatorics](#), involving a re-labeling of the faces of a pair of six-sided dice to reproduce the same frequency of [sums](#) as the standard labeling. The Sicherman dice are crazy dice that are re-labeled with only [positive integers](#). (If the integers need not be positive, to get the same

文 A 7 languages ▾

Read Edit View history Tools ▾

A pair of Sicherman dice. Opposite faces sum to 5 on the left die, and 9 on the right. ▾

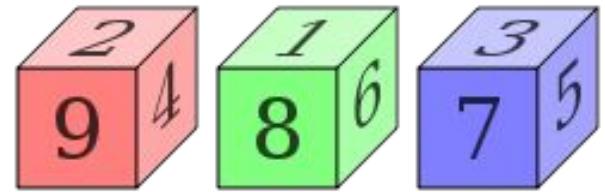
Next Week

How to search for stuff...

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



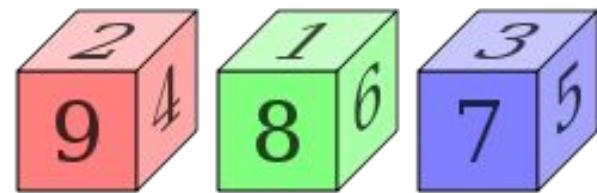
Non-transitive dice:

- A beats B.
- B beats C.
- C beats A
- Choosing second is the best strategy!

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



Questions:

- Algorithm for deciding if dice are non-transitive?
- Can you design a set of 10 non-transitive dice?
- Can you design a set of 20-sided non-transitive dice?