CSCI 161
Introduction to Computer Science
Why Do Cockpits Make Us Nervous?

- Are you sure you know what you're doing?
- What if you press the wrong button?
Outline

- Information Hiding
  - Access Modifiers: public vs private

- Constants: The final Keyword

- Static Keyword
  - Static Variables and Constants
  - Static Methods
  - Life without BlueJ

- Conclusion
Imagine if we're a company that builds **water heaters**.

- Which of the following designs should we market?
Imagine if we're a company that builds water heaters.

- Which of the following designs should we market?

Lesson: Revealing so many elements to the public, we expose too much implementation details.
Anything `public` means **outside** classes can access it

- Seems like it could be a good idea to make your fields `public`...
  - No need to write getters and setters
  - Users know about `public` elements because they are documented in the API

```java
/** @version 1.0 */
public class Student {
   //fields
   public String n;    /** name of student */
   public double g;    /** gpa of student (should be within 0 and 4) */

   /**
    * Constructs a new Student with no name and no GPA
    */
   public Student() {
      ...  
   }
}
```
Peek at the API for Student

Anything labeled public will be published in the API documentation!

```
public class Student extends java.lang.Object

A very poorly designed Student class. Note the fields are public

Version: 1
Author: David

Field Summary

<table>
<thead>
<tr>
<th>Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>double</td>
<td>g</td>
<td>gpa of student (should be within 0 and 4)</td>
</tr>
<tr>
<td>java.lang.String</td>
<td>n</td>
<td>name of student</td>
</tr>
</tbody>
</table>

Constructor Summary

Student()
Constructs a new Student with no name and no GPA

Method Summary

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail
```
Problem 1: Misuse of Fields

- Let's say I didn't comment the `public` fields

- Users of this class might then **guess** at the following:
  - Field `n` is the *neck length*??
  - Field `g` is the *German proficiency score*?
    - Also, don't forget that GPAs can only be a value between 0.0 and 4.0

- Nothing stops users (or even myself) from misusing the fields:

```java
Student david = new Student();
david.n = "Long";       // Uhhh.. maybe n stands for neck length?
david.g = 78.0;         // Ummm.. maybe g stands for German proficiency score?
```
Problem 2: No Backward Compatibility

- My professors tell me that my field names are awful, and aren't representative of the data I'm storing.
  - Okay, I'll **refactor** the class so that my fields have better names.
    - Get everyone to download and upgrade the code for my new class
    - Problems abound! *(Why?)*

```java
/** @version 1 */
public class Student {
    //fields
    public String name; /** name of student */
    public double GPA; /** gpa of student (should be within 0 and 4) */
}
```

Fixed, right? What's the big deal?
Information Hiding with Access Modifiers

- How do we hide implementation details of our classes from users?

- Mechanism: Access modifiers determine the visibility of various elements from external classes:
  - Fields, methods mostly.
  - Even classes can be hidden too. (CS 261)

- There are several access specifiers in Java
  - We can just focus on public and private
  - (There's also package and protected). (CS 261)
Information Hiding with the `private` Specifier

- I don't want users to access or even know about the fields outside this class. Let's hide them.

- Any field or method declared `private` are only visible to elements in the same class.

```java
/** @version 2 */
public class Student {
    //fields
    private String n;  //** name of student */
    private double g;  //** gpa of student (should be within 0 and 4) */
}
```

Now these are hidden from the API (users of this class won't know about their existence!)
Fixing Problem 1: Preventing Misuse

- **Solution:** Minimize opportunities for misuse

- The fields are no longer accessible outside of Student class' code:
  - Yes! That prevents users from tinkering with and misusing a Student object!

```
Student david = new Student();
david.n = "David";
> Error: n has private access in Student

System.out.println("Student's name: " + david.n);
> Error: n has private access in Student
```

- Okay, then **how do we** manipulate and read these fields if they're private?
public class Student {
    //fields
    private String n;    /** name of student */
    private double g;    /** gpa of student (should be within 0 and 4) */

    public String getName() {
        return this.n;
    }

    public double getGPA() {
        return this.g;
    }

    public void changeName(String newName) {
        this.n = newName;
    }

    public void setGPA(double newGPA) {
        //make sure given newGPA is within a legal range
        if (newGPA >= 0.0 && newGPA <= 4.0) {
            this.g = newGPA;
        } else {
            this.g = 0.0;
        }
    }
}

Dealing with Problem 1: Forcing users to use changeName() means they won't confuse this method for "setting the neck length"

Dealing with Problem 1: Forcing users to use setGPA() means we can prevent illegal GPAs to be entered
Peek at Student API

- Field names aren't exposed anymore

```
Constructor Summary

Student()
  Constructs a new Student with no name and no GPA and no gender

Method Summary

java.lang.String getGPA()

java.lang.String getName()

int getRank()

void setGPA(double newGPA)
  Sets the student's GPA.

void setName(java.lang.String newName)
  Sets the name of student

void setRank(int newRank)
  Sets the student's class rank (see constants)
```
Code Example

public class Student {
    //fields
    private String student_name;    /** name of student */
    private double student_gpa;    /** gpa of student (should be within 0 and 4) */

    public String getName() {
        return this.n;
    }

    public double getGPA() {
        return this.g;
    }

    public void changeName(String newName) {
        this.n = newName;
    }

    public void setGPA(double newGPA) {
        //make sure given newGPA is within a legal range
        if (newGPA >= 0.0 && newGPA <= 4.0) {
            this.g = newGPA;
        } else {
            this.g = 0.0;
        }
    }
}

Recall Problem 2: No Backward Compatibility

Let's say I decide to change my field names later on...
public class Student {
    //fields
    private String name;  /** name of student */
    private double gpa;  /** gpa of student (should be within 0 and 4) */

    public String getName() {
        return this.name;
    }

    public double getGPA() {
        return this.gpa;
    }

    public void changeName(String newName) {
        this.name = newName;
    }

    public void setGPA(double newGPA) {
        //make sure given newGPA is within a legal range
        if (newGPA >= 0.0 && newGPA <= 4.0) {
            this.gpa = newGPA;
        }
        else {
            this.gpa = 0.0;
        }
    }

    Dealing with Problem 2:
    I changed my implementation, but because it was hidden from users to begin with, the public API doesn't change!
Peek at Student API (Same as before!)

- Code is much more readable now, yet no change to outsider users!

```
Constructor Summary

**Student()**
Constructs a new Student with no name and no GPA and no gender

Method Summary

<table>
<thead>
<tr>
<th>java.lang.String</th>
<th>getGPA()</th>
</tr>
</thead>
<tbody>
<tr>
<td>java.lang.String</td>
<td>getName()</td>
</tr>
<tr>
<td>int</td>
<td>getRank()</td>
</tr>
<tr>
<td>void</td>
<td>setGPA(double newGPA)</td>
</tr>
<tr>
<td></td>
<td>Sets the student's GPA.</td>
</tr>
<tr>
<td>void</td>
<td>setName(java.lang.String newName)</td>
</tr>
<tr>
<td></td>
<td>Sets the name of student</td>
</tr>
<tr>
<td>void</td>
<td>setRank(int newRank)</td>
</tr>
<tr>
<td></td>
<td>Sets the student's class rank (see constants)</td>
</tr>
</tbody>
</table>
```
Example: Access Modifiers (Inside)

```java
public class Thing {
    public int bizz;
    private int fizz;

    public void foo() {
        // (A)
    }

    private void bar() {
        // (B)
    }
}
```

- From **inside** the class:
  - What **fields** are accessible in (A)?
  - What **methods** are accessible in (A)?
  - What **fields** are accessible in (B)?
  - What **methods** are accessible in (B)?
Example: Access Modifiers (Outside)

- From **outside** the class:
  - Which of Thing's **fields** are accessible from (C)?
  - Which of Thing's **methods** are accessible from (C)?

```java
public class Thing {
    public int bizz;
    private int fizz;

    public void foo() {
        // ...
    }

    private void bar() {
        // ...
    }
}

public class OtherThing {
    public void moo() {
        Thing t = new Thing();
        // (C)
    }
}
```
Summary of Information Hiding

- Relationships of access specifiers

**Java Access Specifiers**

- **public**
  
  Public elements are visible from:
  
  -- **Inside** the class or
  -- **outside** the class

- **private**

  Private elements are visible from:
  
  -- **Inside** the class only!
Outline

- Information Hiding
  - Access Modifiers: public vs private
  - Private Methods
- Constants: The final Keyword
- Static Keyword
  - Static Variables and Constants
  - Static Methods
  - Life without BlueJ
- Conclusion
What about private Methods?

- Methods can be hidden too!
  - Important: *private* methods can only be called by other methods or constructors defined in the *same* class.

- But why would we want to hide methods?
What about private Methods?

- Methods can be hidden too!
  - Important: private methods can only be called by other methods or constructors defined in the same class.

- But why would we want to hide methods?

```java
public class TreeAnalyzer {
    /** Expose just one method */
    public void analyze(String filename) {
        this.openFile(filename);
        this.doLotsOfWork();
    }

    /** Hidden */
    private void openFile(String filename) {
        // ...
    }

    /** Hidden */
    private void doLotsOfWork() {
        // ...
    }
}
```
What about private Methods? (2)

- **Rule of thumb**: Hide a method when...
  - The method is just a "subroutine" for another method to do its job
  - It doesn't make sense for users of your class to call it

- Called *Helper Methods* since they help other methods do their work:
  - Outside world doesn't need to know these "helper methods" exist
  - Letting users call them might break the program
    - *(Just like fields, hiding methods is for users' own protection)*
API: Application Programming Interfaces

- Expose (make public) only the "knobs and levers" we want users to use
  - Hide (make private) implementation details from users

Implementors of a class

Users of a class

Class

```
public pfield
private
private
private
...
```

```
public method0()
public method1()
public method2()
private
private
private
...
```

Class API ("black box")

```
pfield
method0()
method1()
method2()
```

Public API

Use this class by studying API
Summary of Information Hiding (1)

Advantages:

- Minimizes misinterpretation and misuse of code
  - Reduces potential for bugs
- Changes to implementation are abstracted away from users
  - An upgrade on my part doesn't require re-writing code on yours

A bad API:
"How do I use this?"
Caveats?

- Opaqueness of the "black box" model => Users need to have faith in:
  - Good public interface design
  - Good documentation

A good API (blackbox):
"Phew! This is all I need to know?"
Outline

- Information Hiding
  - Access Modifiers: public vs private
- Constants: The final Keyword
- Static Keyword
  - Static Variables
  - Static Methods
  - Life without BlueJ
- Conclusion
Consider a new problem... really long names screw up our roster

Hubert Blaine Wolfeschlegelsteinhausenbergenderoff
David Chiu
Adam Smith
Pablo Diego José Francisco de Paula Juan Nepomuceno María de los Remedios Cipriano de la Santísima Trinidad Ruiz y Picasso
Brad Richards
America Chambers

For better-looking rosters, the university adopts the following rule:

"Truncate student's name to 16 characters"

Hubert Blaine Wo
David Chiu
Adam Smith
Pablo Diego José
Brad Richards
America Chambers

- (Update code in BlueJ now!)
public class Student {
    private String name;
    private double GPA;

    public Student(String name) {
        if (name.length() <= 16) {
            this.name = name;
        }
    }

    public void changeName(String newName) {
        if (newName.length() <= 16) {
            this.name = newName;
        }
    }

    // lots of other methods omitted
Motivation for "final" Values

- But the school keeps changing its mind
  - 1 year later, 32 characters allowed!
  - 3 years later, 40 characters allowed!
  - 5 years later 64 characters allowed!
  - ...

- Problem:
  - Multiple places to change the hard-coded value
    - Every time you change the code, you risk making an error
After Many Years, Things Start Breaking...

```java
public class Student {
    private String name;
    private double GPA;

    public Student(String name) {
        if (name.length() <= 64) {
            this.name = name;
        }
    }

    public void changeName(String newName) {
        if (newName.length() <= 64) {
            this.name = newName;
        }
    }

    // lots of other methods omitted
}
```

Oops! Someone screwed up at one point! (Didn't see these)

Now we spend many more hours debugging what's wrong.
Lesson: NEVER "Hardcode" Values!

public class Student {
    private String name;
    private double GPA;
    private int MAX_NAME_LENGTH = 64;

    public Student(String name) {
        if (name.length() <= MAX_NAME_LENGTH) {
            this.name = name;
        }
    }

    public void changeName(String newName) {
        if (newName.length() <= MAX_NAME_LENGTH) {
            this.name = newName;
        }
    }

    // lots of other methods omitted
}

Now there's only 1 place to make the change.

Errors averted!
New Problem: Outside Classes Want Access

- University class wants to know what the MAX_NAME_LENGTH is too
  - Sure, just make it public.

```java
public class Student {
    private String name;
    private double GPA;
    public int MAX_NAME_LENGTH = 64;
}
```

- But then, this happens:

```java
public class University {
    public void dontKnowWhatImDoing() {
        Student s = new Student(""");

        // I can break the rules for this one student....
        s.MAX_NAME_LENGTH += 40;
        s.changeName("Hubert Blaine Wolfeschlegelsteinhausenbergerdorff .......");
    }
}
```
Solution: The **final** Keyword

- **MAX_NAME_LENGTH** is a constant! Don't allow it to change!

```java
public class Student {
    private String name;
    private double GPA;
    public final int MAX_NAME_LENGTH = 64;  // can't be changed later!

    public Student(String name) {
        if (name.length() <= MAX_NAME_LENGTH) {
            this.name = name;
        }
    }

    public void changeName(String newName) {
        if (newName.length() <= MAX_NAME_LENGTH) {
            this.name = newName;
        }
    }
}
```
Now this won't even compile (Good!)

```java
public class University {
    public void dontKnowWhatImDoing() {
        Student s = new Student("";

        // I can break the rules for this one student....
        s.MAX_NAME_LENGTH += 10;  // Drat! This won't compile
        s.changeName("kdlfkalsjfkjdlfkajfkdjsflkjsdfjdfjdf")
    }
}
```
Important Dates:

- Homework 7 (Pump) due Friday!
- Homework 8 pitch also due Friday
  - Upload text proposal to Canvas

Today:

- Static keyword
- What is main() all about?
- Other code editors: IntelliJ IDEA
- Start analysis
Lab 10 Post-Mortem (Election Time!)
  • Due tonight -- Finish Part 1

Hwk 7 (pump)
  • Due tonight

Last time...
  • Finished HashMap and Files
  • Best Practices: Why not public fields?

Today:
  • Information hiding
  • Final keyword and constants