CSCI 161
Introduction to Computer Science
Outline

- Information Hiding
  - Access Modifiers: public vs private

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

- Conclusion
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

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

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

Version: 1
Author: David

Anything labeled public will be published in the API documentation!

### 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

```java
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 (No Defensive Control over Field Values)

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

- Users of this class must 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 = 95.0;   //Ummm.. maybe g stands for German proficiency score?
```

- Need a way to control what gets assigned!
Problem 2: No Backward Compatibility

- My friends 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?
Imagine if the engineer built the water tank with all its guts exposed, and told you to tinker with it as you please.

- Things would break very quickly because I would misuse *something* I didn't understand.

Analogous Problem: By making everything public, we expose *too much* implementation details.
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 though. (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!

```java
Student david = new Student();
david.n = "long";
> Error: n has private access in Student
david.n = "David"
> Error: n has private access in Student
david.g = 95.0;
> Error: g 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 n;
    }

    public double getGPA() {
        return g;
    }

    public void setName(String newName) {
        n = newName;
    }

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

- Field names aren't exposed anymore

<table>
<thead>
<tr>
<th>Constructor Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Student()</td>
</tr>
<tr>
<td>Constructs a new Student with no name and no GPA and no gender</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Method Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>java.lang.String getGPA()</td>
</tr>
<tr>
<td>java.lang.String getName()</td>
</tr>
<tr>
<td>int getRank()</td>
</tr>
<tr>
<td>void setGPA(double newGPA)</td>
</tr>
<tr>
<td>Sets the student's GPA.</td>
</tr>
<tr>
<td>void setName(java.lang.String newName)</td>
</tr>
<tr>
<td>Sets the name of student</td>
</tr>
<tr>
<td>void setRank(int newRank)</td>
</tr>
<tr>
<td>Sets the student's class rank (see constants)</td>
</tr>
</tbody>
</table>
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 n;
  }

  public double getGPA() {
    return g;
  }

  public void setName(String newName) {
    n = newName;
  }

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

Recall Problem 2: No Backward Compatibility

Let's say I decide to change my field names later on...
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, but no change to outsider users!
  - Even after I made significant changes to implementation

```java
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)
```
Outline

- Information Hiding
  - Access Modifiers: public vs private
  - Private methods

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

- Conclusion
What about private Methods?

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

- Usually used when the private method is simply *helping other* methods which I expose to the outside world.
  - Users don't need to know these "helper methods" exist

- Rule of thumb - Hide a method when...
  - It doesn't make sense for users of your class to call it
  - The method adds confusion about how to use objects of your class
public class ChatBot {
    private Random rng;
    private ArrayList<String> responses;

    public void start() {
        printWelcome();
        Scanner keyboard = new Scanner(System.in);  // Used to get keyboard input
        String user_input;  // Stores their response
        do {
            user_input = keyboard.nextLine();  // Get user input
            if (!user_input.equalsIgnoreCase("bye")) {
                printSnarkyResponse();
            }
        } while (!user_input.equalsIgnoreCase("bye"));
        printGoodbye();
    }

    private void fillSnarkyResponses() {
        responses.add("Say, do you like cats?");
        responses.add("No one has ever complained about this before.");
        responses.add("You're not making any sense. Could you ask in a different way?");
        responses.add("I just Googled it. It doesn't know either.");
        responses.add("Calm down, I don't want to argue.");
    }

    private void printWelcome() {
        System.out.println("Tech support! How can I help?");
        System.out.println("Type 'bye' to end this session.");
    }
}
Outline

▪ Information Hiding
  • Access Modifiers: public vs private

▪ Static Keyword
  • Static Fields and Constants
  • Static Methods
  • Life without BlueJ

▪ Conclusion
Static (Class) Variables

- We know about **fields (or instance variables)**
  - Variables remembering things about objects
- Consider this problem:
  - We want to keep track of how many Students have been created
Why Won't This Work?

Let's consider what happens when we create some Student objects

```java
public class Student {
    //fields
    private String name;
    private double GPA;
    private int numStudents;

    //constructors
    public Student() {
        //inside constructor... this means a student was created
        numStudents++;
    }

    //remaining methods omitted
}
```

- Let's consider what happens when we create some Student objects
Declare numStudents as a Class Variable

```java
public class Student {
    //class (static) variables
    private static int numStudents;

    //fields
    private String name;
    private double GPA;

    //constructors
    public Student() {
        //inside constructor... this means a student was created
        Student.numStudents++;
    }

    //remaining methods omitted
}
```

- If `public`, any elements outside of class can read (or write to) it
- If `private`, then only the class elements can read (or write to) it
public class A {

    // static vars
    private static double u;
    public static double v;

    // instance vars
    private double x;
    public double y;

    public void m1() {
        // does this code compile?
        x = 1;
        y = 2;
        A.u = 3;
        A.v = 4;
    }
}

public class A {

    // static vars
    private static double u;
    public static double v;

    // instance vars
    private double x;
    public double y;
}

public class B {

    // which of the following assignments are invalid?
    public void m2() {
        A obj = new A();
        //obj.x = 1;
        //obj.y = 2;
        //A.u = 3;
        A.v = 4;
    }
}
Outline

- Information Hiding
  - Access Modifiers: public vs private

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

- Conclusion
### Constants

- One popular use for static variables is to define *constants*
  - Constants are read-only variables; their values *cannot* be changed
    - Use the `final` keyword to define constants

- Some class constants you might've used already:
  - `Math.PI` (approx. value of $\pi$)
  - `Double.NaN` (not a number)
  - `Double.POSITIVE_INFINITY`, `Double.NEGATIVE_INFINITY`

  **Notice the Classname.varName syntax in all these constants!**

- We've been using these in *our own* classes
  - What does that say about the *visibility* of these constants?
Where Constants Are Useful

- Suppose we want to add a class `rank` to our Student class:
  - Freshman, Sophomore, Junior, and Senior

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

    // (constructors and methods omitted)

    public void setRank(String newRank) {
        rank = newRank;
    }

    public String getRank() {
        return rank;
    }
}
```
Create a class, University

- Write a method that checks if a student is a senior, and prints an invitation to a graduation event if they are.

```java
public class University {
    // fields
    private ArrayList<Student> allStudents;

    // constructors and methods omitted

    public void printInvitations() {
        // loop through the list of all students
        for (int i = 0; i < allStudents.size(); i++) {
            // check if they're a senior
            if (allStudents.get(i).getRank().equalsIgnoreCase("senior")) {
                System.out.println("Dear " + allStudents.get(i).getName() + ",");
                System.out.println("You're invited to the HACK-athon!");
            }
        }
    }
}
```

Focus on this check. Looks to see if the student's rank is "senior"
Where Constants are Useful (Cont.)

- Bad data entry would cause `printInvitations` to break!

- Example:

```java
Student david = new Student();
david.setName("David");
david.setGPA(3.0);
david.setRank("Senior");

Student hans = new Student();
hans.setName("Hans");
hans.setGPA(3.5);
hans.setRank("Sr."); // this is fine right?

Student taylor = new Student();
taylor.setName("Taylor");
taylor.setGPA(3.8);
taylor.setRank("SR"); // SR also means senior

University ups = new University();
ups.addStudent(david);
ups.addStudent(hans);
ups.addStudent(taylor);
ups.printInvitations(); // who gets the invitation?
```

**Problem:** It's hard to enforce consistency when we accept any `String` as rank.
Define some constants. For instance: `Student.SENIOR` will always have a value of 4.

```java
public class Student {
    //constants
    public static final int FRESHMAN = 1;
    public static final int SOPHOMORE = 2;
    public static final int JUNIOR = 3;
    public static final int SENIOR = 4;

    //class variables
    private static int numStudents;

    //fields
    private String name;
    private int rank;
    private double GPA;

    //some constructors and methods omitted
    public void setRank(int newRank) {
        rank = newRank;
    }

    public int getRank() {
        return rank;
    }
}
```

Ranks are `ints` now, not `Strings` anymore.
public class University
{
    //fields
    private ArrayList<Student> allStudents;

    //constructors and methods omitted
    public void printInvitations()
    {
        //loop through the list of all students
        for (int i = 0; i < allStudents.size(); i++)
        {
            //check if they're a senior
            if (allStudents.get(i).getRank() == Student.SENIOR)
            {
                System.out.println("Dear " + allStudents.get(i).getName() + ",");
                System.out.println("You're invited to Logger Bash!");
                System.out.println("Be there or be square.");
            }
        }
    }
}
Bad Data Entry Avoided

- Users now use constants to set rank

- Example:

```java
Student david = new Student();
david.setName("David");
david.setGPA(3.0);
david.setRank(Student.SENIOR);

Student hans = new Student();
hans.setName("Hans");
hans.setGPA(3.5);
hans.setRank(Student.SENIOR);

Student taylor = new Student();
taylor.setName("Taylor");
taylor.setGPA(3.8);
taylor.setRank(Student.SENIOR);

University ups = new University();
ups.addStudent(david);
ups.addStudent(hans);
ups.addStudent(taylor);
ups.printInvitations();  //now the right people get the invitation
```
Outline

- Information Hiding
  - Access Modifiers: public vs private
- Static Keyword
  - Static Variables and Constants
  - Static Methods
  - Life without BlueJ
- Conclusion
(TOTALLY FAKE) Example for Motivation of Static Methods:

```java
// we want to compute the hypotenuse...
double base = 50;
double height = 60;

//okay gotta create a Math object
Math mathObj = new Math();
double hypotenuse = mathObj.sqrt(mathObj.pow(base, 2) + mathObj.pow(height, 2));

//now I have to print
PrintStream out = new PrintStream();
out.println("The hypotenuse is: " + hypotenuse);
```
A **class (static) method** allows us to call it without creating any objects!

- **Important**: Static methods can't access each object's fields directly

Usually defined when a method is providing a popular utility not limited to your class.

Again, we've used *static methods* in the past...

- Math.sqrt()
- Math.pow()
- System.out.println()
- Integer.parseInt()

Notice the `Classname.method()` syntax!
(TOTALLY FAKE) Example for Motivation of Static Methods:

```java
// we want to compute the hypotenuse...
double base = 50;
double height = 60;

// ughhh.. gotta create a Math object now..
Math mathObj = new Math();
double hypotenuse = mathObj.sqrt(mathObj.pow(base, 2) + mathObj.pow(height, 2));

// now I have to print... ugggggggg construct another object!
PrintStream out = new PrintStream();
out.println("The hypotenuse is: "+ hypotenuse);
```

Better (Why create objects to call these common methods!?)

```java
// we want to compute the hypotenuse...
double base = 50;
double height = 60;
double hypotenuse = Math.sqrt(Math.pow(base, 2) + Math.pow(height, 2));

// now I have to print
System.out.println("The hypotenuse is: "+ hypotenuse);
```
Example: Temperature Class

```java
public class Temperature {

    // constants
    public static final int BOILING_C = 100;  /** boiling point in celsius */
    public static final int FREEZING_C = 0;   /** freezing point in celsius */
    public static final int BOILING_F = 212;  /** boiling point in fahrenheit */
    public static final int FREEZING_F = 32;  /** freezing point in fahrenheit */

    // static methods
    /**
     * @return the converted temperature from Celsius to Fahrenheit
     */
    public static double celsiusToFahrenheit(double degrees) {
        return 9.0/5.0 * degrees + FREEZING_F;
    }

    /**
     * @return the converted temperature from Fahrenheit to Celsius
     */
    public static double fahrenheitToCelsius(double degrees) {
        return 5.0/9.0 * (degrees - FREEZING_F);
    }
}
```
Example of Calling Static Methods:

```java
// we want to know what 23.5 is in Fahrenheit
Temperature.fahrenheitToCelsius(23.5)
> 74.30000000000001 (double)

Temperature.celsiusToFahrenheit(0.0)
> 32.0 (double)

Temperature.fahrenheitToCelsius(212.0)
> 100.0 (double)
```
When to Write Static Methods?

- Pro-tip: When to make a method static?
  - Ask yourself: Should I let users call it, even if it an object of the class is not created?

```java
Temperature.celsiusToFahrenheit(0.0) > 32.0 (double)
Temperature.fahrenheitToCelsius(212.0) > 100.0 (double)
```

```java
System.out.println("Hello World!");
```

```java
double x = Math.log(1024) / Math.log(2); // log (base 2) of 1024
```
Recall, code units (methods, fields, ...) declared static exist in the class scope, and there's only one copy of them.

**Class Scope**

![Class Scope Diagram]

**Object (Instance) Scope**

![Object (Instance) Scope Diagram]
Rule: Static code cannot directly access code in object scope.

- Can: create objects using constructor, and call methods on object

Class Scope

Object (Instance) Scope

Static code has no access to its object's fields and methods
Rule: Object code can access static elements

- Objects do have access to static fields and methods.
Scoping Example

- Compile errors exist in this code. Identify them.

```java
public class Bugs {
    /** static scope */
    public static int val = 1;

    public static void foo() {
        System.out.println(val);
        System.out.println(val2);
    }

    /** object (instance) scope */
    private String val2;

    public Bugs() {
        foo();
    }

    public int foo() {
        Bugs.val++;
        return Bugs.val;
    }
}
```
Compile errors exist in this code. Identify them.

```java
public class Bugs {
    /** static scope */
    public static int val = 1;

    public static void foo() {
        System.out.println(val);
        System.out.println(val2);
    }

    /** object (instance) scope */
    private String val2;

    public Bugs() {
        foo();
    }

    public int foo() {
        Bugs.val++;
        return Bugs.val;
    }
}
```

- Static methods can't access its own object's fields.
- Method name already defined, even though it's defined in a different scope!
- This is okay. Static variables are accessible from object code.
Outline

- Information Hiding
  - Access Modifiers: public vs private
  - Helper Classes

- Static Keyword
  - Static Fields and Constants
  - Static Methods
    - Life without BlueJ
      - The main() Method

- Conclusion
Life with BlueJ

To get your programs to run:

- Step 1: You code up some classes in BlueJ
- Step 2: To run your program, you either:
  - Type stuff into CodePad, or
  - Create objects on the Workbench, then run its methods from the Workbench
Life *without* BlueJ

- Not how it works in real life...
  - BlueJ is a CS1 educational tool
  - You won't (shouldn't) use BlueJ in CS 261 and beyond

- Programmers use a fully Integrated Development Environment (IDE)
  - Examples include:
    - Eclipse, IntelliJ, NetBeans (for Java programs)
    - Android Studio (for Android)
    - Microsoft Visual Studios (for C#)

<<IntelliJ demo>>
The main() Method

- Would be nice if you can run a "script" that contains all the object interactions
  - Key observation: You have been acting out this script from within BlueJ's codepad and workbench!

- To run programs without BlueJ, every project must have at least one main() method:
  - Important: Must have this exact signature:

```java
public static void main(String[] args) {
    //do stuff you would've done inside codepad
}
```
How to Use main()

- Step 1: Open the project in IntelliJ
- Step 2: Optionally create a new class, which has only the main() method defined.
  - You can also put main() inside one of the current classes
- Step 3: Write the code inside main()
  - It should create objects
  - Boss them around to perform your goal
- Step 4: Run it!
Outline

- Information Hiding
  - Access Modifiers: public vs private

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

- Conclusion
Homework 7 (Pump) due Friday!

Homework 8 pitch also due Friday
  • Via email!
Administrivia 4/12

- Important Dates:
  - Homework 7 (Pump) pushed back to next Friday!
  - Homework 8 pitch also due next Friday
    - Upload text proposal to Canvas

- Today:
  - Information hiding
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