Topic 10.0: Interfaces

© Lauren Himbeault 2024

Learning Goals:

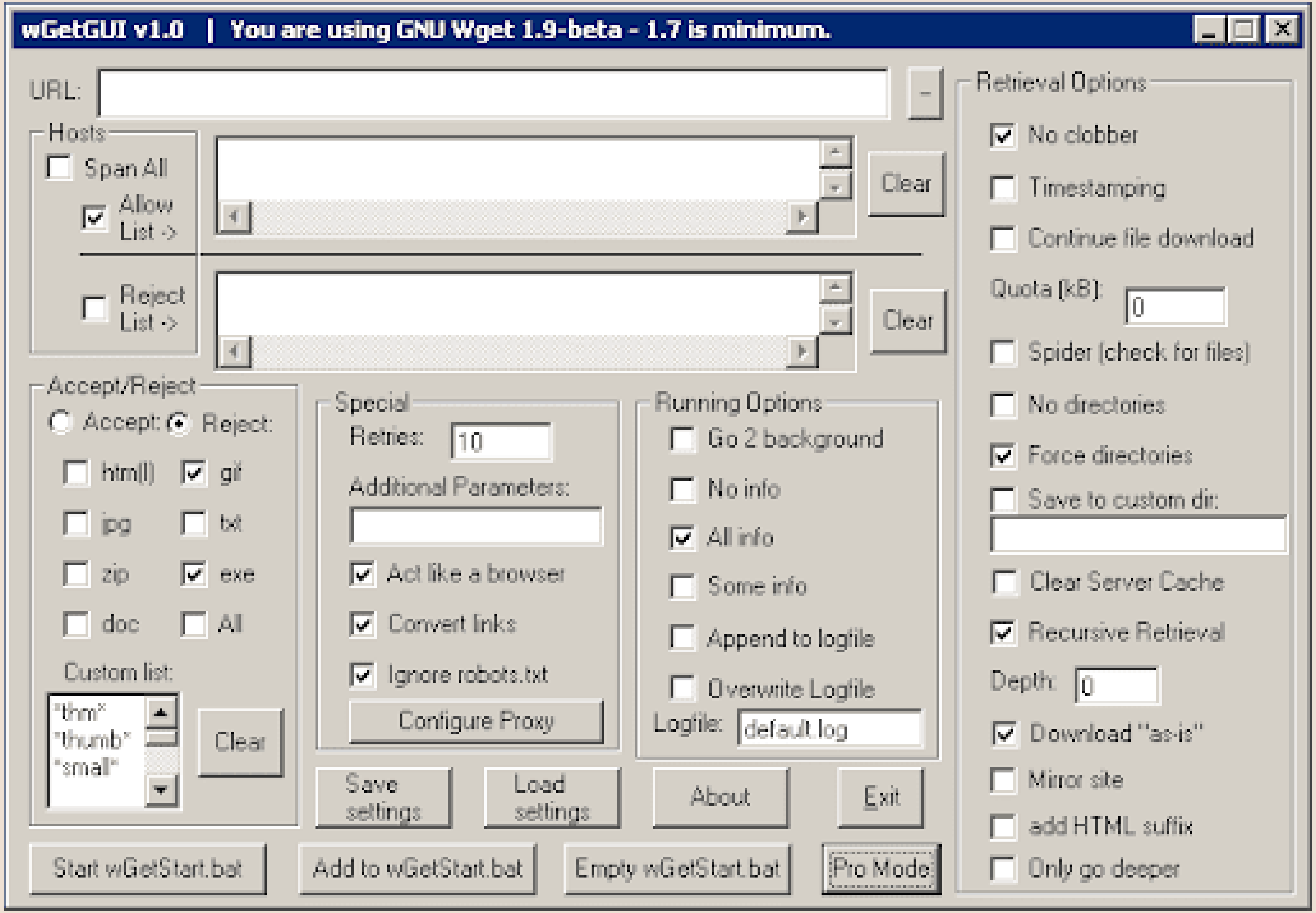
- Differentiate between an interface and its implementation.
- Force a class to implement abstract methods by having it implement an interface.
- Use interfaces as variable types, parameter types, and return value types.

© Lauren Himbeault 2024

Interfaces

- User Interface? (**Not** the kind of interface we are talking about **but** we can use it as an intro)
  - what we interact with
  - the space where we interact with technology
  - usually has a nice layout, controls, etc.
- A **good user interface** describes (often without instructions) what we **can** do with a piece of technology/machine
- A **bad user interface** might produce unexpected/surprising results, frustrating execution paths, or incorrect outcomes

© Lauren Himbeault 2024



wGetGUI v1.0 | You are using GNU Wget 1.9-beta - 1.7 is minimum.

URL:

Hosts
Span All
[✓ Allow List ->]
Reject List ->

Clear
Clear

Accept/Reject
Accept: [✓ Reject:]
htm(l) [✓ gif]
jpg txt
zip [✓ exe]
doc All
Custom list:
thm
thumb
small
Clear

Special
Retries: 10
Additional Parameters:
[✓ Act like a browser]
[✓ Convert links]
[✓ Ignore robots.txt]
Configure Proxy

Running Options
Go 2 background
No info
[✓ All info]
Some info
Append to logfile
Overwrite Logfile
Logfile: default.log

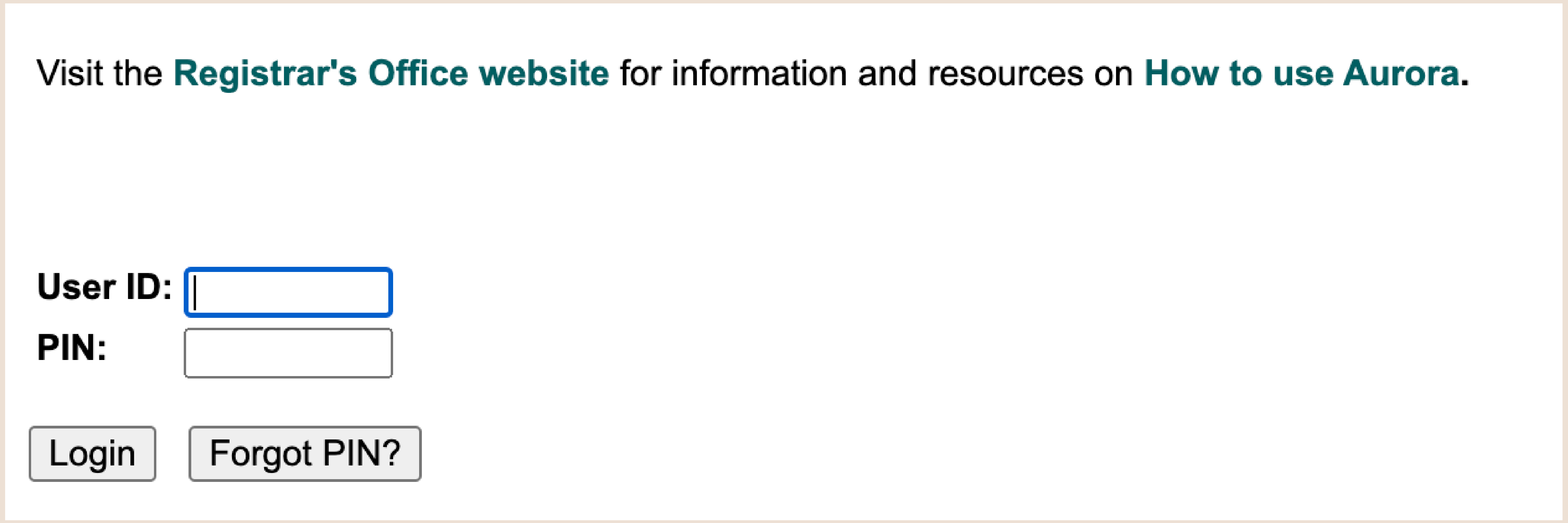
Retrieval Options
[✓ No clobber]
Timestamping
Continue file download
Quota (kB): 0
Spider (check for files)
No directories
[✓ Force directories]
Save to custom dir:
Clear Server Cache
[✓ Recursive Retrieval]
Depth: 0
[✓ Download "as-is"]
Mirror site
add HTML suffix
Only go deeper

Save settings
Load settings
About
Exit

Start wGetStart.bat
Add to wGetStart.bat
Empty wGetStart.bat
Pro Mode

http://justaddwater.dk/2006/12/06/worst-user-interface-ever/

Interfaces

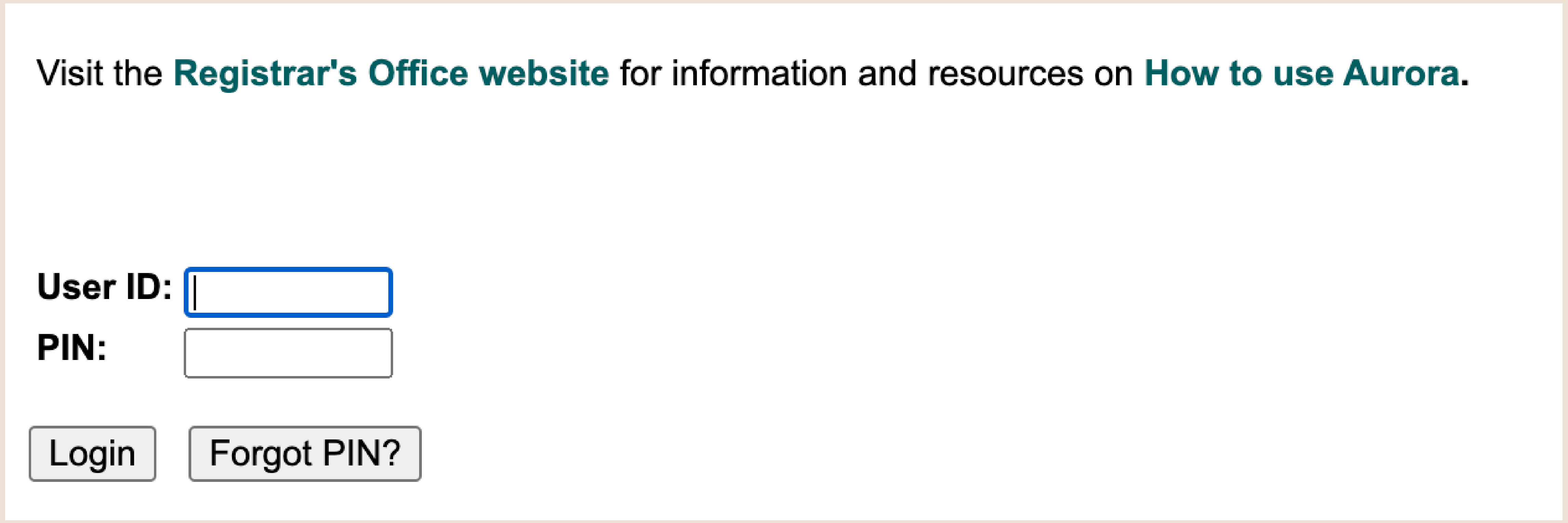

Visit the Registrar's Office website for information and resources on How to use Aurora.

User ID:
PIN:

- User Interface
  - How is our data being stored? **Don’t know**
  - How is our information being process? **No Idea**
  - What data structures are being used? **Don’t have that information**
  - How many methods get called to make a ‘submit’ happen? **Not sure**

© Lauren Himbeault 2024

Interfaces


Visit the Registrar's Office website for information and resources on How to use Aurora.

As a User of the UI...

I DON'T CARE!

- User Interface
  - How is our data being stored? **Don't know**
  - How is our information being process? **No Idea**
  - What data structures are being used? **Don't have that information**
  - How many methods get called to make a 'submit' happen? **Not sure**

© Lauren Himbeault 2024

Interfaces: The Coding Kind

- A Java class **encapsulates** (remember that word?) the data that defines it, and the operations that we can perform on it.

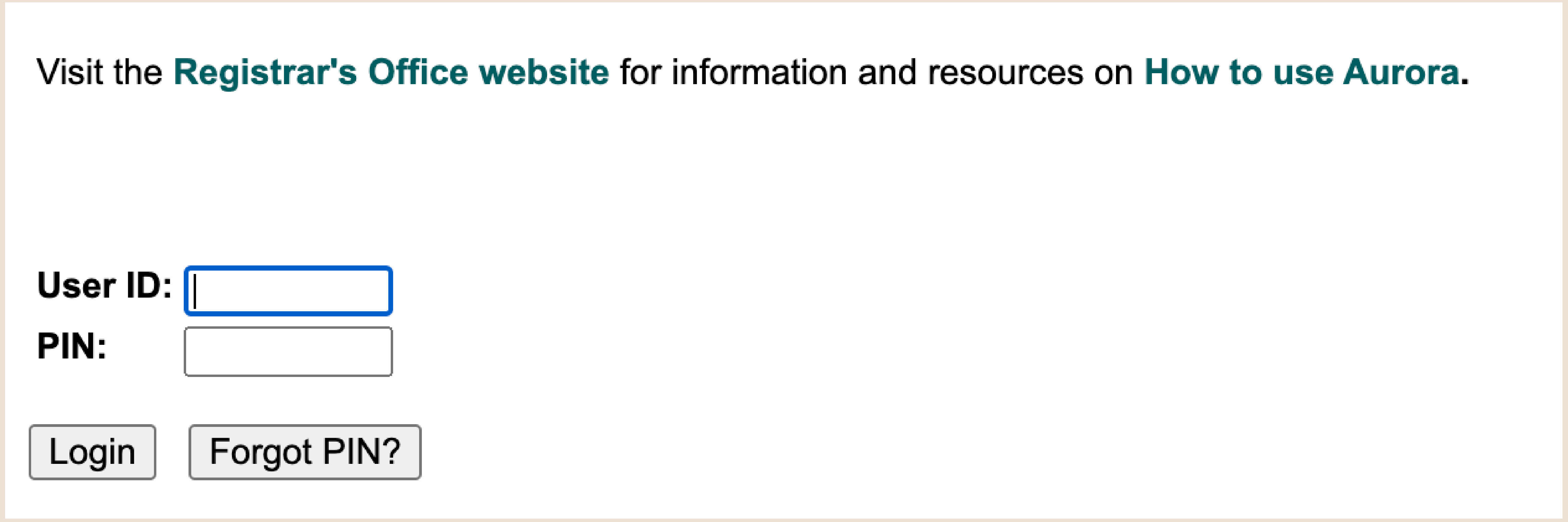
- The data **(instance variables)** define its properties; that is, what a particular object is.
- The operations **(instance methods)** define how it processes that data.

- But even more important is a definition of what it **can** do.

© Lauren Himbeault 2024

Interfaces

- What **can** I do here?


Visit the **Registrar's Office website** for information and resources on **How to use Aurora**.

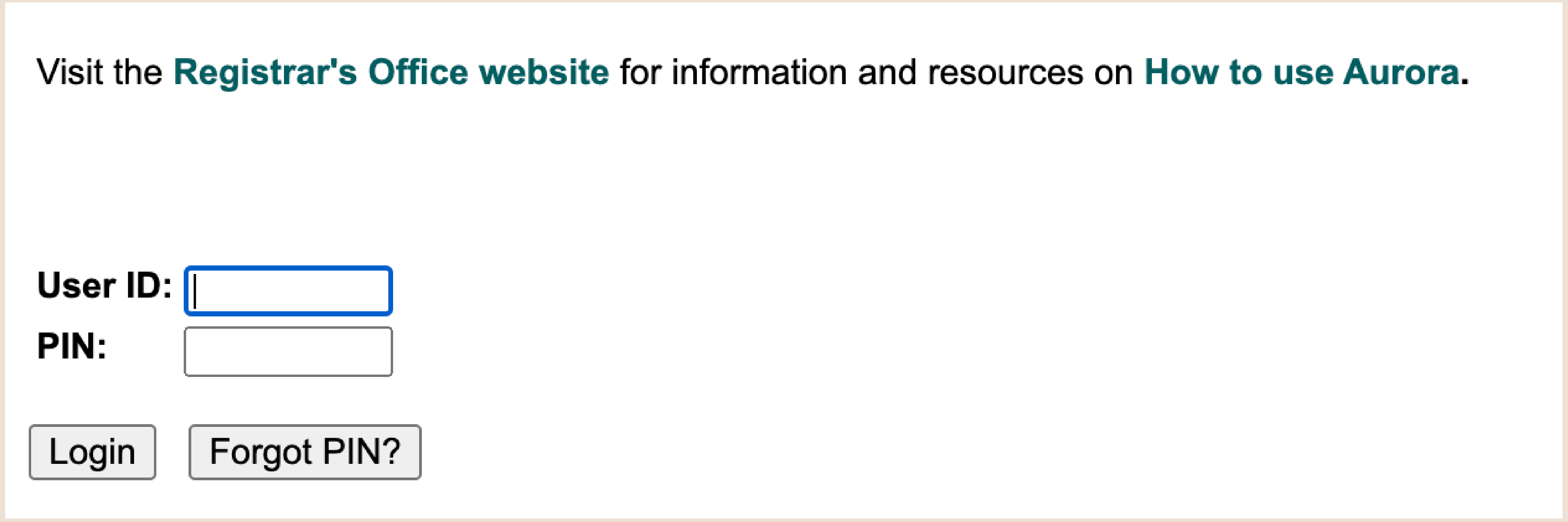
User ID:
PIN:

- Enter a User ID
- Enter a PIN
- Click Log In
- Click Forgot PIN

© Lauren Himbeault 2024

Interfaces

- What **can** I do here?


Visit the Registrar's Office website for information and resources on How to use Aurora.

User of the UI only cares about what **they can do with the interface**

- Enter a User ID
- Enter a PIN
- Click Log In
- Click Forgot PIN

The developer can then decide **how to do those things**

© Lauren Himbeault 2024

Interfaces: The Coding Kind

- We can do this in our code!
- We can define an **interface** that outlines **behaviours** and then lots of classes can use the interface and define **how those behaviours are implemented**

- **BARE WITH ME!**
    - Initially, this will seem like an extra step to doing what our code ALREADY ca do but I **promise** to show you examples of where it is helpful

© Lauren Himbeault 2024

Interfaces: The Coding Kind

- The **public part** of our classes is the only way that other code can use our classes.

- That is, the headers of all our public methods are how the main() or other pieces of code interact with our classes

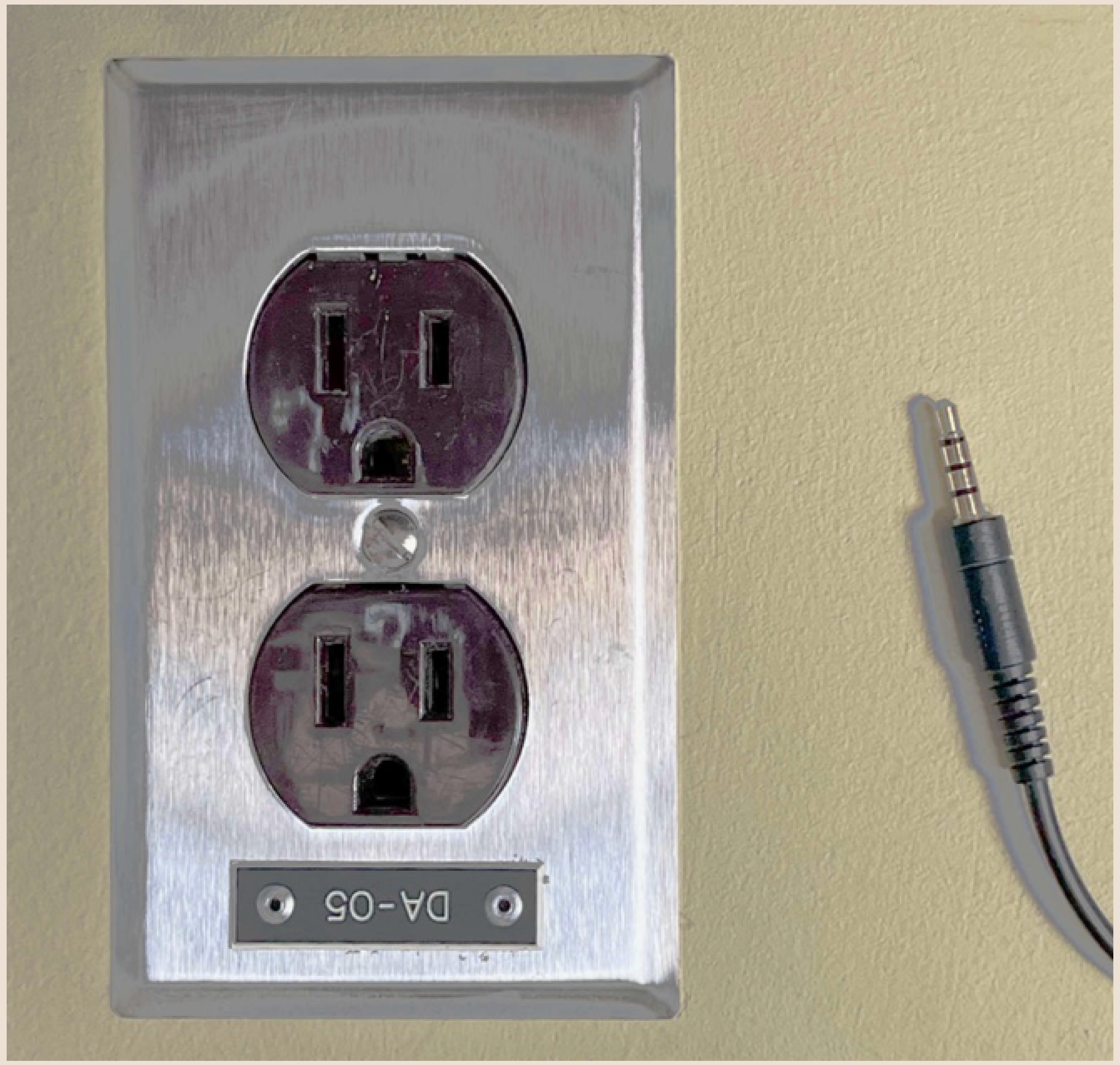
- **These method headers define the interface to our class.** They define what our class can do.

- Without these, nobody could use our class.
- They are what provide access to the rest of the program.
    -

© Lauren Himbeault 2024

Interfaces:

- Interfaces are not just a digital space thing either
- In the real world, we use interfaces everywhere. They restrict access to things, often with good reason



Interfaces can ensure that we don't misuse something beyond its intended purpose.

© Lauren Himbeault 2024

Interfaces vs Implementation

- Ideally the “public” parts of our classes are the only parts that the rest of our program gets to see.
- The other parts, like instance variables, and the code that makes up our methods, are hidden. (we make them private right now)
- These other parts (private) of our class are known as its implementation.

© Lauren Himbeault 2024

Interfaces vs Implementation

- Ideally the “public” parts of our classes are the only parts that the rest of our program gets to see.

- The other parts, like instance variables, and the code that makes up our methods, are hidden. (we make them private right now)

- These other parts (private) of our class are known as its implementation.

- Consider the method
    - public String toString() { // toString things }
- Do we know how the data inside is stored? Do we know anything about the implementation? **Hint: NO**
- All we know is when we call this method, we will get a String of the data from thge instance object

© Lauren Himbeault 2024

Hiding Implementation

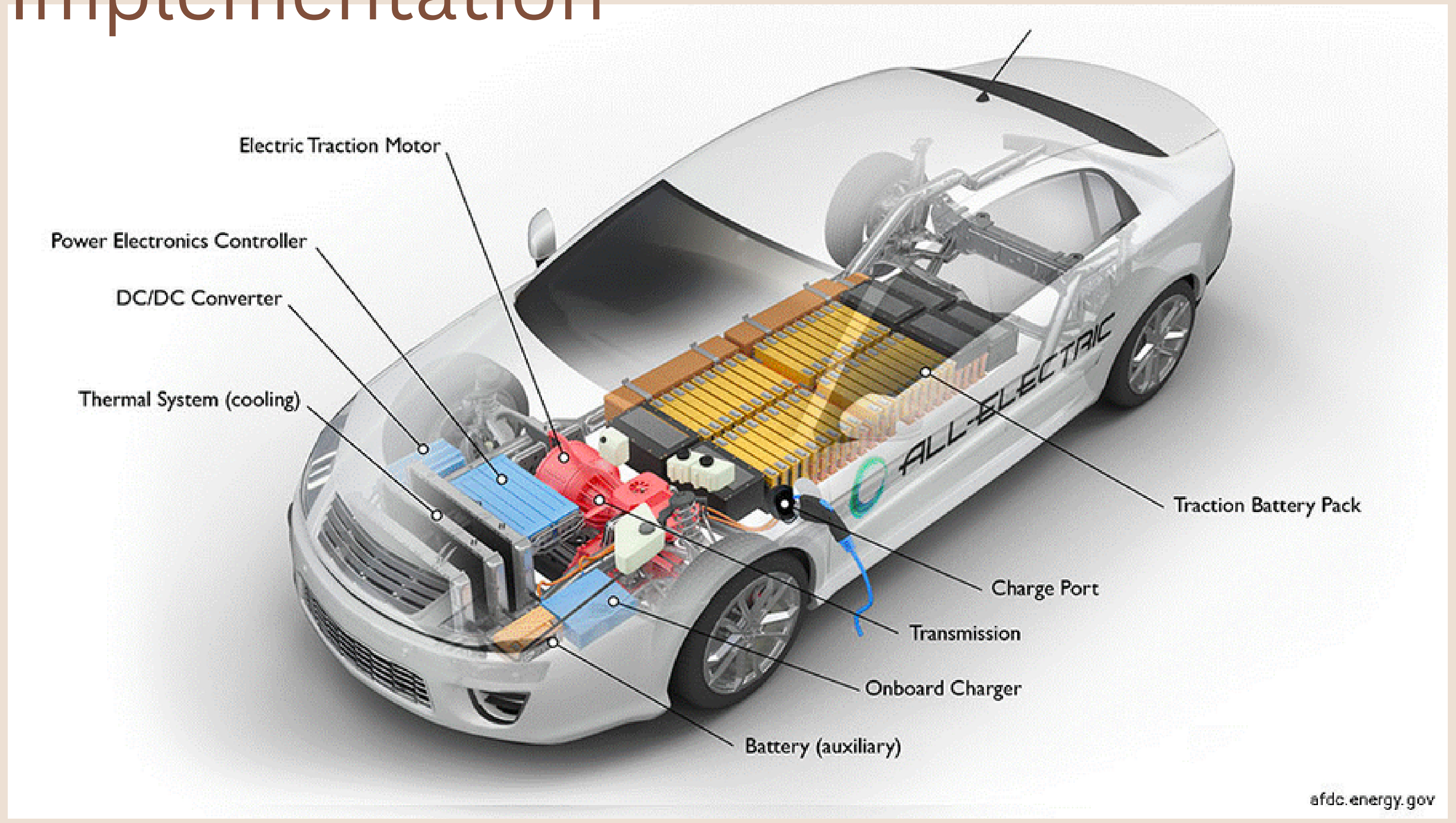
- We separate (**conceptually**) the public interface from the private implementation.

- This enables us to use the class without having to know how it works.
- This separation is critical when our projects get larger and more complex!
    - promise!

Interface

shutterstock

Implementation



© Lauren Himbeault 2024

A Simple Example

- Here’s a method from a class that stores a list of university courses:

```
// Here’s the public part:
public class CourseList {



    public void sortByCourseName() {



    }
}
```

© Lauren Himbeault 2024

A Simple Example

- Here’s a method from a class that stores a list of university courses:

```
// Here’s the public part:
public class CourseList {
        // How we store the list of courses is private
        // Array? ArrayList? LinkedList? Something else?

    public void sortByCourseName() {

        // The sorting algorithm we use is private too
        // Selection? Insertion? Bubble? Something else?

    }
}
```

© Lauren Himbeault 2024

A Simple Example

- The interface forms a wall that protects the privacy of our implementation from the rest of the program. We can change the sort, or the list, without changing the part of the program that uses it.

- courseList.sortByCourseName(); // doesn’t matter how it happens, we just know the list is sorted by course name after this method call

© Lauren Himbeault 2024

A Simple Example

- The interface forms a wall that protects the privacy of our implementation from the rest of the program. We can change the sort, or the list, without changing the part of the program that uses it.

- courseList.sortByCourseName(); // doesn’t matter how it happens, we just know the list is sorted by course name after this method call
- We can use the CourseList in a large program without knowing the details like how the courses are stored, or how we sort them.
  - This enables us to treat the entire class as an independent subproblem of a larger problem.
  - We can assume it works as advertised: it does what the interface says it does.
  - In the future, it will enable us to work with others on a team.

© Lauren Himbeault 2024

A BIG Example: APIs

- An API, or **Application Programming Interface**, defines an interface that is published for different programs to use.

- You have been using the Java API since you first printed “Hello, World”.

- You can call System.out.println from the Java API without knowing how it works!
  - You just know that after you use it, you get some stuff on your console

https://docs.oracle.com/en/java/javase/11/docs/api/

© Lauren Himbeault 2024

Formal Java Interfaces

- Java allows you to formally **specify an interface, separately from the class that implements it.**

- A Java **interface** (this is a key word in Java) lists public method headers but doesn’t include private stuff (no implementations)

- Just a list of public method headers followed by semicolons.
- Writing our interface separate from our implementation means:
  - We can write an interface and then one or more classes that **implements** (another Java key word) it.
  -

© Lauren Himbeault 2024

Example: ITablePrintable

- This interface describes methods that will neatly print the contents of a class in a table:

```
public interface ITablePrintable {
    void printHeader();
    void printData();
}
```

- Note that method headers omit the word “public”: all the methods in our interface are implicitly public.
- There are no instance variables or method code.
- See the TablePrinter.zip folder for the full example

© Lauren Himbeault 2024

Example: ITablePrintable

- What are some things that might have data we want to print in a table?
  - Courses?

- Course Header should print out:

  Faculty: Department

- Course Data should print out (for each):

  CourseCode: CourseName   Instructor: InstructorName

- See the TablePrinter.zip folder for the full example (using CourseMain.java to run)

© Lauren Himbeault 2024

Example: ITablePrintable

- What are some things that might have data we want to print in a table?
  - Courses?
  - **Students?**

- Student Header should print out:

  Student Name: Student Number

- Student Data should print out (for each):

  CourseCode: CourseName   Grade: GradeInClass

- See the TablePrinter.zip folder for the full example (using StudentMain.java to run)

© Lauren Himbeault 2024

Some Important Notes

- The methods listed in the interface **must have identical headers in every class that implements them.**
  - Except they **must be explicitly public.**
  - A class **must implement all of them.**
-
- A class can implement more than one interface (comma separated)
  - e.g.:
    - class Course implements TablePrinter, Registration { ...

© Lauren Himbeault 2024

So now what?

- Both class can implement the interface....cool.....
- We could have achieved this by just making those two methods in the two classes **without** the **implements** in the class header and it would still work
- All we've done is successfully written more code that added ZERO value to our program
- **What's the catch....?**

© Lauren Himbeault 2024

Interface POWER

- Both class can implement the interface….cool…..
- We could have achieved this by just making those two methods in the two classes **without** the **implements** in the class header and it would still work
- All we've done is successfully written more code that added ZERO value to our program
- **What's the catch….?** WELL I'LL TELL YOU!

- You can declare variables of an interface type, and have them refer to any object whose class implements that interface
  - See InterfacePowerMain.java

© Lauren Himbeault 2024

Interface POWER

- **Whoa.**

- There is a catch to this though:
    - we cannot call .getCourseNumber() on the ITablePrintable object
        - **even if it is a Course in there**
    - In fact, you can't call any method that's not listed in the interface through the variable of the interface type (at least, not directly; more later)

© Lauren Himbeault 2024

Interface POWER

- **More power!**

- You can use them as parameters:

```
public static void printInTable(TablePrinter object) {
     // more fancy printing code not shown
     object.printHeader();
     object.printData();
}

public static void main(String[] args) {
     // make course and student variables
     printInTable(course);
     printInTable(student);
}
```

© Lauren Himbeault 2024

Interface POWER

- **LISTS**
- We can use an interface type to describe the kind of object we store in a collection.

```
// works just as well with arrays
ArrayList<TablePrinter> printable = new ArrayList<>();

printable.add(course);
printable.add(student);

for (TablePrinter p : printable) {
     p.printData();
}
```

© Lauren Himbeault 2024

Pause & Practice

- Create an interface named **IMakesSound**, and then implement this interface in several animal classes (Dog, Cat, Mouse). Use the provided AnimalInterfaceMain.java to test your implementations
- The Animal classes you make have a **String name** and have a **default constructor**
- In the constructor, set the name to the name of the class
- IMakesSound should have one defined method header, void makeSound() which should print the sound that animal makes to the console
- In your classes, you can implement the makeSound() however you want

© Lauren Himbeault 2024