CITS4401 Software Requirements and Design
Interfaces and system design

Lecturer: Rachel Cardell-Oliver
Notes: Arran Stewart

Interfaces

What is an *interface*?

Interfaces

What is an *interface*?

It just means, “the boundary where two things meet”.

Summary of Interfaces Topic

1. How are interfaces defined? say how services are requested and how are they delivered
2. Types of interfaces:
   human to computer (user interface)
   hardware
   software
3. Application Programmer Interfaces
4. System design with interfaces: Top down or Bottom up

Interfaces

In software engineering, we'll usually be interested in situations where at least one of the things provides some sort of *service* to the other.

The interface will be *defined* by

- how and in what format those services are requested
  - including, what must be *provided* when requesting a service, and
  - any *conditions* that must be satisfied
- how and in what format the services are delivered
  - including anything that will be true *after* they've been delivered

Example – DoT

We could talk about the interface between a *customer* and the *Driver and Vehicle Services* section of the WA Department of Transport, that lets you request a driver's license renewal.

- How do you request this service?

Example – DoT

We could talk about the interface between a *customer* and the *Driver and Vehicle Services* section of the WA Department of Transport, that lets you request a driver's license renewal.

- How do you request this service?
  - There are multiple ways, but one is to *attend a DoT office in person*.

Example – DoT

We could talk about the interface between a *customer* and the *Driver and Vehicle Services* section of the WA Department of Transport, that lets you request a driver's license renewal.

- How do you request this service?
  - There are multiple ways, but one is to *attend a DoT office in person*.
- What do you need to provide?

Example – DoT

We could talk about the interface between a *customer* and the *Driver and Vehicle Services* section of the WA Department of Transport, that lets you request a driver's license renewal.

- How do you request this service?
  - There are multiple ways, but one is to *attend a DoT office in person*.
- What do you need to provide?
  - You need to fill in a paper form with information about your existing license.

Example – DoT

We could talk about the interface between a *customer* and the *Driver and Vehicle Services* section of the WA Department of Transport, that lets you request a driver's license renewal.

- How do you request this service?
  - There are multiple ways, but one is to *attend a DoT office in person*.
- What do you need to provide?
  - You need to fill in a paper form with information about your existing license.
- Are there any conditions that need to be satisfied, before the DoT will provide the service?

Example – DoT

We could talk about the interface between a *customer* and the *Driver and Vehicle Services* section of the WA Department of Transport, that lets you request a driver's license renewal.

- How do you request this service?
  - There are multiple ways, but one is to *attend a DoT office in person*.
- What do you need to provide?
  - You need to fill in a paper form with information about your existing license.
- Are there any conditions that need to be satisfied, before the DoT will provide the service?
  - Yes! You need to actually have an existing driver's license.

Example – DoT

We could talk about the interface between a *customer* and the *Driver and Vehicle Services* section of the WA Department of Transport, that lets you request a driver's license renewal.

- How do you request this service?
  - There are multiple ways, but one is to *attend a DoT office in person*.
- What do you need to provide?
  - You need to fill in a paper form with information about your existing license.
- Are there any conditions that need to be satisfied, before the DoT will provide the service?
  - Yes! You need to actually have an existing driver's license.
  - And you can't be disqualified (e.g. due to traffic offences committed)

Example – DoT

We could talk about the interface between a *customer* and the *Driver and Vehicle Services* section of the WA Department of Transport, that lets you request a driver's license renewal.

- How do you request this service?
  - There are multiple ways, but one is to *attend a DoT office in person*.
- What do you need to provide?
  - You need to fill in a paper form with information about your existing license.
- Are there any conditions that need to be satisfied, before the DoT will provide the service?
  - Yes! You need to actually have an existing driver's license.
  - And you can't be disqualified (e.g. due to traffic offences committed)
  - And you may have to satisfy some other conditions as well (e.g. proof of eyesight)

Example – DoT

How does the DoT provide the service?

- They will issue you immediately with a paper *temporary* license

Example – DoT

How does the DoT provide the service?

- They will issue you immediately with a paper *temporary* license
- And in the mail, they will send you a new physical driver's license.

Inerfaces in SE

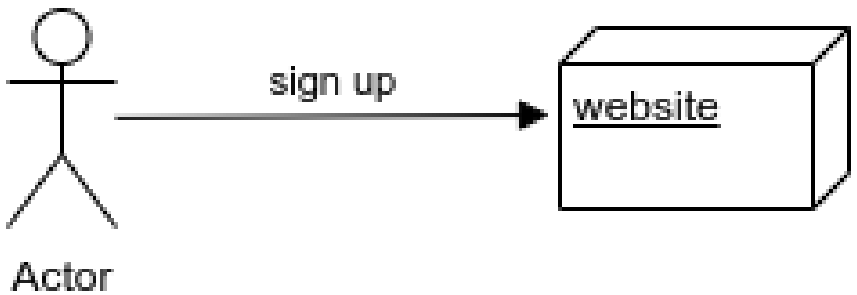
In software engineering, we'll usually be interested in the case where at least *one* of the two things involved – usually the thing providing the service – is a computer artifact of some sort (an item of hardware or software).

The thing *requesting* the service might also be a computer artifact, but could be human.

And we will typically want the answers to the questions given in the previous slides to be *clearly defined*.

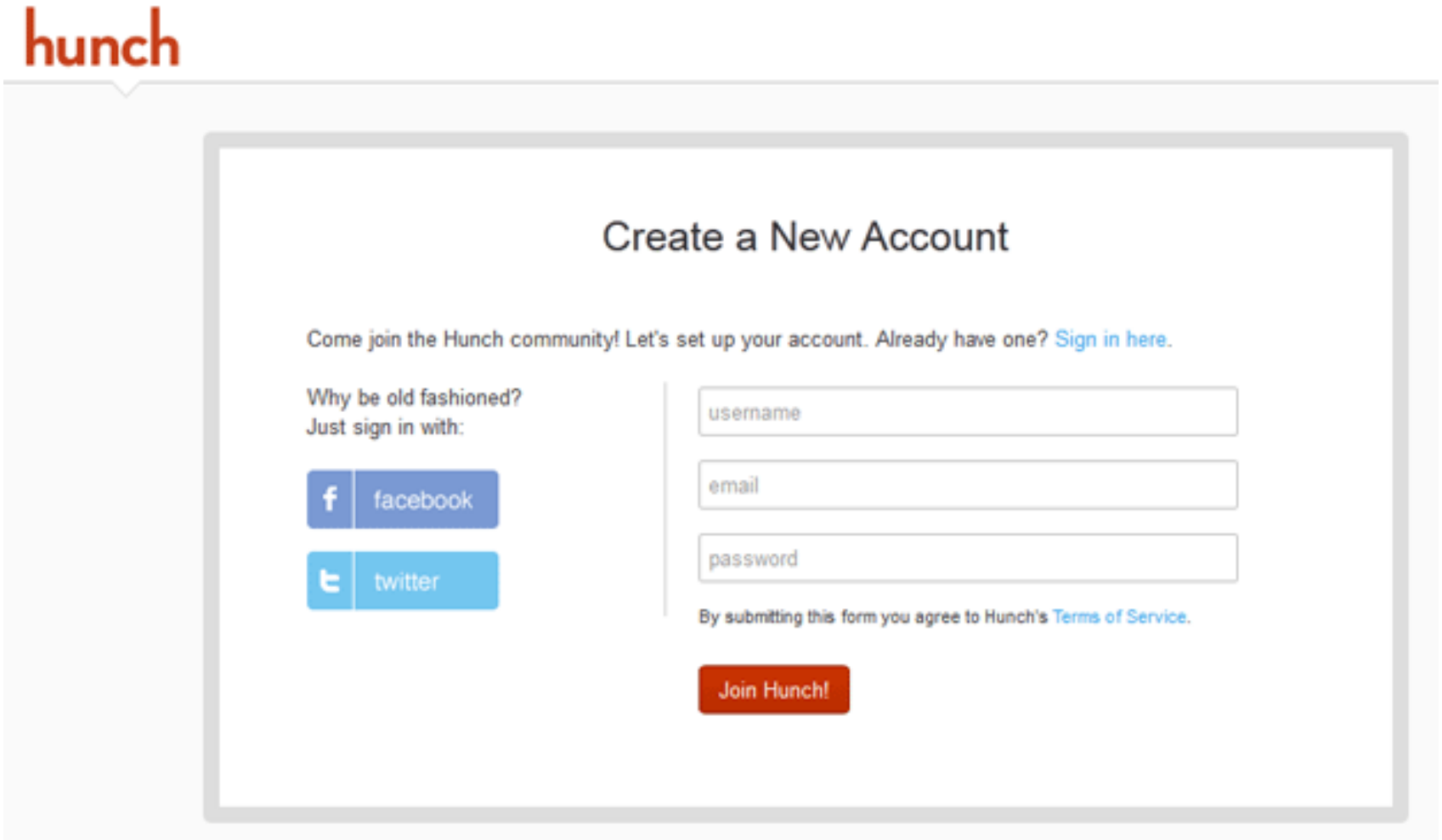
User interfaces

If the thing requesting the service is a person – a *user* – then the interface is a *user interface*.



User interfaces

Here's an example of a user interface:



User interfaces

- How does a user make the request?

User interfaces

- How does a user make the request?
  - They either click one of the buttons on the left, *or*

User interfaces

- How does a user make the request?
  - They either click one of the buttons on the left, *or*
  - they fill in the text fields, and click the button underneath

User interfaces

- How does a user make the request?
  - They either click one of the buttons on the left, *or*
  - they fill in the text fields, and click the button underneath
- Are there any conditions that must be satisfied in order for the user to request the service?

User interfaces

- How does a user make the request?
  - They either click one of the buttons on the left, *or*
  - they fill in the text fields, and click the button underneath
- Are there any conditions that must be satisfied in order for the user to request the service?
  - Often, the email address must be in a valid form. (Additional validation might get performed later – e.g. by sending a confirmation email – but it's not a precondition for using the form.)

User interfaces

- How is the service delivered?

User interfaces

- How is the service delivered?
  - The “service” just consists in the fact that after having signed up, the user will *have an account*.

Hardware interfaces

If both of the things involved are items of *hardware*, then the service will be specified both in terms of physical connections, and perhaps what sort of signals are passed through it and what they mean.

Hardware interfaces

To specify the hardware interface of the parallel port in the previous slide, we'll specify

- the physical characteristics of the port – exactly what dimensions does it have, and what dimensions and configuration must an appropriate plug have to fit into it?
- often – what signals can be sent across each of the wires the port attaches to, and what they mean.

Summary

1. How are interfaces defined? say how services are requested and how are they delivered
2. Types of interfaces:
   human to computer (user interface)
   hardware
   software
3. Application Programmer Interfaces
4. System design with interfaces: Top down or Bottom up

Interfaces to program artifacts

Software has interfaces to – well-defined ways of requesting some bit of software perform a service. The software could be:

- a method
- a class
- a set of classes

(Also, Java uses the word "interface" to refer to a class-like thing which specifes an interface, but not an implementation.)

APIs

If we look up the Java documentation for the `String.indexOf()` method, we are told the following:

Its signature is:

```
int indexOf(int ch)
```

That is, it takes an `int`[1] and returns an `int`.

[1] Why not a char? For historical reasons we won't get into.

APIs

And also, we are told what this method does:

> Returns the index within this string of the first occurrence of the specified character. If a character with value ch occurs in the character sequence represented by this String object, then the index (in Unicode code units) of the first such occurrence is returned. For values of ch in the range from 0 to 0xFFFF (inclusive), this is the smallest value k such that:
> `this.charAt(k) == ch`
> is true. For other values of ch, it is the smallest value k such that: `this.codePointAt(k) == ch` is true. In either case, if no such character occurs in this string, then -1 is returned.
>
> **Parameters:**
> ch - a character (Unicode code point).
>
> **Returns:**
> the index of the first occurrence of the character in the character sequence represented by this object, or -1 if the character does not occur.

APIs

Because it's a method, the way to request a service from the method should be pretty clear to all Java programmers – we *call* the method.

```
String s = "banana";
int res = s.indexOf('a'); // using the service
System.out.println(res);
```

In this case, the service is provided to us in the form of a return value. But other methods (e.g. `.println()`) might provide their services by *doing* something (printing things to the screen, erasing our hard-disk, launching intercontinental ballistic missiles, etc.).

APIs

If the thing we're interested in is a *class*, then it can have both public and private members. We are usually interested in the *public* interface – the set of all public methods and instance variables.

Modules

If we have a module – a collection of classes – then

- the specification for all the externally accessible classes, methods and so on in a module make up what is called the *API* of the module – the "Application Programming Interface".
- See further "Who invented the API?", https://nordicapis.com/who-invented-the-api/ Interesting: "Right now, we are in a very interesting place when it comes to the legal questions surrounding APIs"

APIs

If we have a module – a collection of classes – then

- The name derives from the idea that if we write a re-usable component of some sort (like a library), then other developers will want to use this in their application programming, and we should document the public *interface* to that component.
- (Actually, the other developers might not be writing an *application* per se – they might be writing another library – but the name has stuck.)

APIs as contracts

- We can think of the API for a function (or other procedural unit) as constituting a *contract* between the developer of the function, and the client code using it.[2]
- In effect, the documentation says "If you, the client code, pass me arguments which meet the following criteria, I promise to do the following thing: . . . "

[2] See further "Design by contract", https://en.wikipedia.org/wiki/Design_by_contract; Meyer, Bertrand. "Applying 'design by contract'." Computer 25.10 (1992): 40-51.

APIs, cont'd

- The "following thing" – the behaviour of the function – will usually be to return some sort of value, or to cause some sort of "side effect".
- (A *side effect* is anything the function does to alter the current or subsequent behaviour of the system or its interaction with external systems, other than returning a value.
  For example, writing a file to disk, or sending an email, or changing the value of a global variable.)

API example

- If you have used Java, you most likely at some point will have written something like
  `System.out.println("some string ...")`

API example

- If you have used Java, you most likely at some point will have written something like
  `System.out.println("some string ...")`
- What does the `println` method promise to do?

API example

- If you have used Java, you most likely at some point will have written something like
  `System.out.println("some string ...")`
- What does the `println` method promise to do?
- The Javadoc says:

API example

- If you have used Java, you most likely at some point will have written something like
  `System.out.println("some string ...")`
- What does the `println` method promise to do?
- The Javadoc says:

  `void println(String x)`
  Prints a String and then terminate the line.

API example

- If you have used Java, you most likely at some point will have written something like
  `System.out.println("some string ...")`
- What does the `println` method promise to do?
- The Javadoc says:

  `void println(String x)`
  Prints a String and then terminate the line.

- Does `println()` return a value? If not, then what does it promise to do?

APIs – specification vs implementation

The API documentation does not normally say *how* the function is to be implemented – just what its return value and effects are.

This means that if the library developer decides to reimplement the function in another way (for instance, to improve efficiency), they can, without changing the API.

Specification vs implementation example

In fact, you can have multiple implementations of the same API by different developers.

Example:

- Oracle corporation provides an implementation of the Java standard libraries (as well as of the Java compiler, `javac`, and the Java Virtual Machine or JVM).
- But there are other implementations – for instance, OpenJDK, an open-source version of the standard libraries.
- These adhere to exactly the same specifications as the Oracle versions.
- (In fact, since Java version 7, the OpenJDK version has been the reference implementation.)

Specification vs implementation – other examples

- The POSIX standard specifies an API for Unix-like systems, and has been implemented multiple times in different ways by different operating systems. (In fact, even Windows, at various times, has met the POSIX standards.)

Specification vs implementation in Java

- Q. In Java, does the API tell us *how* `String.indexOf(ch)` is implemented? How would you implement it?

Specification vs implementation in Java

- Q. In Java, does the API tell us *how* `String.indexOf(ch)` is implemented? How would you implement it?
- A. It does not. The *plausible* way to do it is to test each possible index from 0 to (length-of-string - 1), see if matches the character we're looking for, and if it does, return the index we're currently at.

Specification vs implementation in Java

- Q. In Java, does the API tell us *how* `String.indexOf(ch)` is implemented? How would you implement it?
- A. It does not. The *plausible* way to do it is to test each possible index from 0 to (length-of-string - 1), see if matches the character we're looking for, and if it does, return the index we're currently at.
- But there's nothing in the *specification* to stop us from implementing it in other ways . . .

Specification vs implementation in Java

- Q. In Java, does the API tell us *how* `String.indexOf(ch)` is implemented? How would you implement it?
- A. It does not. The *plausible* way to do it is to test each possible index from 0 to (length-of-string - 1), see if matches the character we're looking for, and if it does, return the index we're currently at.
- But there's nothing in the *specification* to stop us from implementing it in other ways ...
- e.g. Generate a random number from 0 to (length-of-string - 1), call it k. Check and see if we've hit all positions from 0 to $k-1$ yet; if we have, and `inputString.charAt(k)` equals the character we're after, return the current index.[3]

[3] See also Bogosort, https://en.wikipedia.org/wiki/Bogosort

Specification vs implementation – “illities”

Sometimes specifications for units will describe not just what the unit *returns* or *does*, but *how it does it*.

For instance, if we implement `String.indexOf(ch)` in the “generate a random number” way we described, it would be *extremely* slow.

Specification vs implementation – "illities"

The specification of `String.indexOf(ch)` could rule out "silly" implementations like this, by saying something like "the `indexOf` method shall provide guaranteed $O(n)$ time cost, where n is the length of the string".

(If you have not done a data structures and algorithms course, don't worry too much about what "$O(n)$" means – it roughly means that the time to execute `indexOf` will increase in proportion to the length of the string.)

Specification vs implementation – “illities”

If you look at the documentation for Java’s `TreeMap` class, in fact, you will see that the implementation provided by Oracle promises to provide guarantees about how long particular methods will take to run:

> *This implementation provides guaranteed log(n) time cost for the `containsKey`, `get`, `put` and `remove` operations. Algorithms are adaptations of those in Cormen, Leiserson, and Rivest’s Introduction to Algorithms.*

APIs, summary

So:

- The API describes the *expected behaviour* of a module (or larger system).
- The code constitutes a particular *implementation* of that API.

APIs, cont'd

What should go in the API documentation?

- The *preconditions* – any conditions which should be satisfied by the parameters or the system state when the function is called.
- The *postconditions* – the return value of the function, and any *changes* the function makes to the system state (the "side effects" discussed earlier)

We often also will document what will happen if the preconditions *aren't* satisfied: in many languages, this will typically be an exception being thrown.

APIs, cont'd

Once we know the preconditions and postconditions for a function, we can write tests for it.

(They needn't be spelled out formally or mathematically – but it is best if they are clear, consistent and unambiguous.)

Developing systems – top-down vs bottom-up

When developing systems there are basically two ways of doing it:

- Top-down, also called "stepwise refinement":
  - Try and break the desired system down into smaller pieces, until we have something we can handle
- Bottom-up:
  - Try and identify individual bits of functionality we can identify, and try to assemble them into a system.

Historically, top-down was once more common. (Well – in practice, projects used a mix of the two.) But in OO systems, something more like bottom-up is more common.

Developing systems – top-down vs bottom-up

Therefore, in an OO system – often, when we reach the stage of design, we will *already have* a bunch of classes representing functionality we want.

Sometimes, we will need to break those classes down further, because it's not clear how to implement them.

But often, a more important task is to *group classes together* into components and subsystems.

Summary of Interfaces Topic

1. How are interfaces defined? say how services are requested and how are they delivered
2. Types of interfaces:
   human to computer (user interface)
   hardware
   software
3. Application Programmer Interfaces
4. System design with interfaces: Top down or Bottom up