**1. Introduction to Interfaces**

**Goal**:

* Understand what interface design is, including Human-Computer Interaction (HCI), hardware interfaces, software interfaces, and Application Programming Interfaces (APIs).
* Learn system design with interfaces, using top-down or bottom-up approaches.

**Definition**:

* An interface is the boundary where two things meet. It defines how different components of a system interact with each other.

**2. User Interfaces**

**Definition**:

* A user interface is where a person interacts with a system.

**Examples**:

1. **Text or Button-Based Interface**:
   * **Request**: Users make requests by clicking buttons or filling in text fields and submitting forms.
   * **Conditions**: Input validation, such as a valid email format.
   * **Delivery**: Users receive feedback, such as account creation confirmation.
2. **Voice-Based Interface**:
   * **Request**: Users use voice commands (e.g., "Hey Siri, order me a taxi").
   * **Conditions**: Authentication, language recognition.
   * **Delivery**: The system processes the request and provides feedback through voice responses.
3. **Gesture-Based Interface**:
   * **Request**: Users perform gestures on a touch screen (e.g., swiping, pinching).
   * **Conditions**: Device must be active and permissions granted.
   * **Delivery**: The system detects and interprets gestures to navigate or execute actions.

**3. Hardware Interfaces**

**Definition**:

* Hardware interfaces involve physical connections and signals between hardware components.

**Example**:

* **Parallel Port**:
  + **Physical Characteristics**: Dimensions and configuration of the port and plug.
  + **Signals**: Types of signals transmitted across the wires and their meanings.

**4. Software Interfaces (APIs)**

**Definition**:

* An API defines a set of rules and protocols for how software components interact. It allows different software applications to communicate and perform services for each other.

**Components**:

1. **API Definition**:
   * Specifies methods, endpoints, parameters, and data formats.
   * Example: **GET /weather?city=New York HTTP/1.1**
2. **API Request**:
   * The calling program sends a request to the API, specifying the desired action.
   * Example: An application requests the weather data for a specific city.
3. **Service Execution**:
   * The target program processes the request and performs the operation.
   * Example: The weather service retrieves the data for New York.
4. **API Response**:
   * The target program sends back a response with the results.
   * Example: **{"city": "New York", "temperature": "22°C", "description": "Sunny"}**
5. **Response Handling**:
   * The calling program processes the response.
   * Example: Displaying the weather information to the user.

**Types of Program Artifacts**:

1. **Method**: Defined by the method's signature and functionality (e.g., **calculateTotal()** in a shopping cart class).
2. **Class**: Public methods and properties form the interface (e.g., **Car** class with **startEngine()** and **drive()** methods).
3. **Set of Classes (Subsystems)**: Related classes providing a specific functionality (e.g., account management in a banking application).
4. **External Service**: APIs for web services, databases, or third-party libraries (e.g., weather API).

**Examples**:

* **Google Maps API**: Provides geolocation and mapping services.
* **OpenAI API**: Offers AI services like natural language processing.

**API Notes**:

* APIs act as a contract between developers and client code.
* Specify what the function does, its return value, and effects, but not how it is implemented.
* Multiple implementations can exist, adhering to the same specifications (e.g., different implementations of Java libraries by Oracle and OpenJDK).

**5. Developing Systems with Interfaces: Top-down vs. Bottom-up**

**Top-down (Stepwise Refinement)**:

* Break down the system into smaller pieces until manageable.
* Common in traditional systems design.

**Bottom-up**:

* Identify and assemble individual functionalities into a system.
* More common in object-oriented (OO) systems.
  + 在 OO 系统中，可以对单个类和对象进行单元测试。自下而上的方法有助于在将每个单元集成到更大的系统之前对其进行彻底测试
  + 自下而上的方法允许开发人员首先创建这些对象

**Approaches**:

* **Top-down**: Start with the big picture and refine it.
* **Bottom-up**: Start with specific functionalities and integrate them