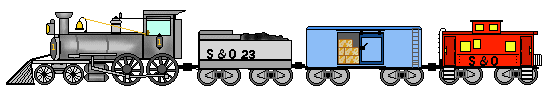
**Physical Linking**

* Let’s consider a real-life example of linking:
  + Pretend you have a container.
  + Inside this container is a random item with a label that says “item” on it.
  + Now imagine this container has a chain connecting it to another container that also has an item in it.
  + And imagine that container is connected to another container, and so on in a uniform line.
  + These containers and chains can be abstracted into code.
* Indeed, this example is similar to how a train works.
  + A train has a head, which is connected to several containers, each of which are holding something, and are connected.
  + Notice how the last container connects to nothing



* Let’s consider how these containers can be seen in code.

**Programmatically Linking**

**Node Abstraction**

* Each container is called a **node**.
* Each node contains two pieces of information.

1. **A data item.**
2. **A pointer to another node.**

* Since each node needs to contain multiple pieces of information, **each node is an object**.
* Each node is linked either linked to
  + **nothing (nullptr)**

or

* + **to another node**

Graphical user interface

Description automatically generated

* Here, the data portion of a node contains a string.
  + It can be any data type (primitive or user defined)
* We use arrows to represent the pointers
  + Each pointer in a node should point to another node (not the data type held within a node)
  + Pointers can point to any data type except files, so this is an important distinction that a nodes pointer should point to another node

**Node Code**

struct Node {

string item;

Node \*next;

};

**Defining a Node**

* To declare a pointer to a node in Java:

**Node ptr;**

* This defines a pointer variable **ptr** that can point to a node of type Node.
* **Nodes should be dynamically allocated.**
* To initialize a node in Java:

**ptr = new Node();**

* This allocates a node to which **ptr** points.
* To access the members of a node, you will need the . notation.