CS 2100: Data Structures & Algorithms 1

List, Interfaces, and Polymorphism

Dr. Nada Basit // basit@virginia.edu

Spring 2022

Friendly Reminders

- Masks are **required** at all times during class (University Policy)
- If you forget your mask (or mask is lost/broken), I have a few available
  - Just come up to me at the start of class and ask!
- No eating or drinking in the classroom, please
- Our lectures will be **recorded** (see Collab) – please allow 24-48 hrs to post
- If you feel **unwell**, or think you are, please stay home
  - *We will work with you!*
  - At home: eye mask instead! Get some rest ☺

What is a List?

Java Collections Framework (JCF)

- Java has several frameworks, one of them is called "Collections" **– Java Collections Framework**
- These are classes and libraries, etc that support "**containers**" for storing items

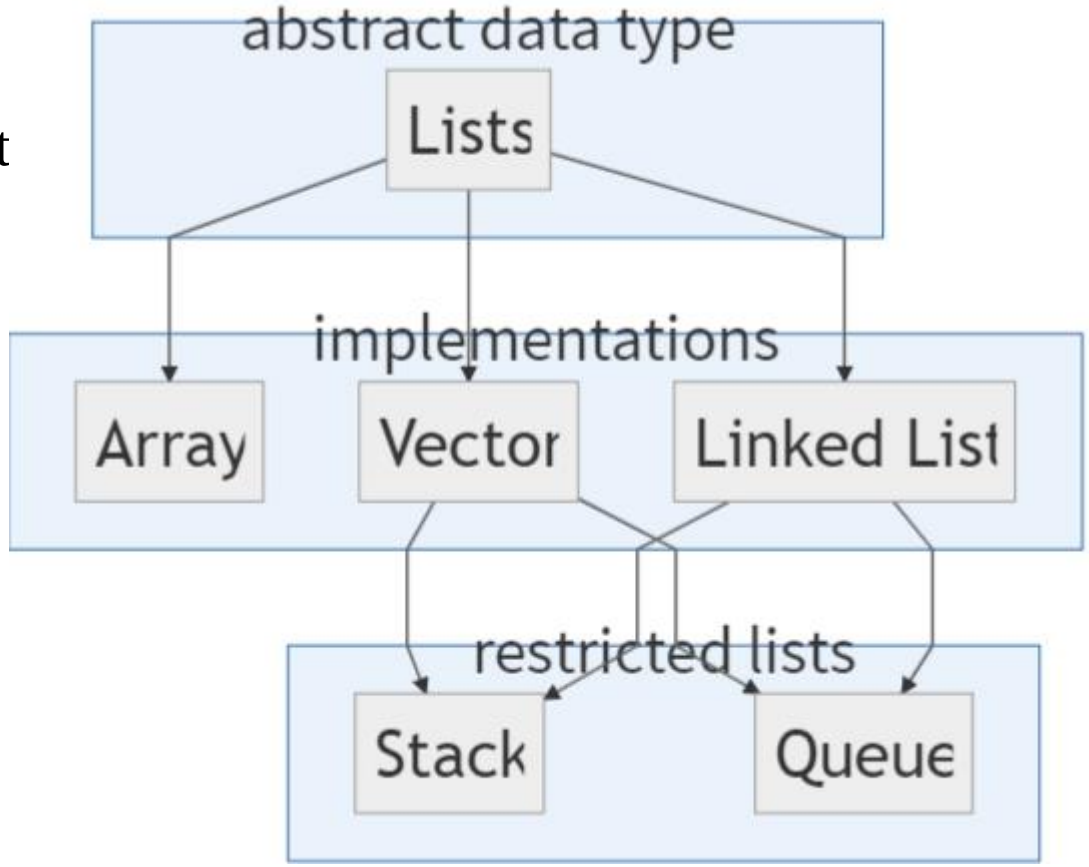
- The JCF has three (3) fundamental types:
  - **List**: stores objects in order (just like arrays)
  - **Set**: Stores unique set of objects (no duplicates)
  - **Map**: Stores key/value pairs (like Python dictionaries)
- Choose the right container to match your application!

What is a List?

- A **data model** that **maintains order** as items are added
- A collection of items (**Arrays** are an example):
  - **Indexed** in order from 0 to n-1
  - Allows for positional access (access anywhere in the list via the index)
  - Can use standard for-loop or for-each loop to traverse the structure
  - Can **add** things to the list, **remove** things, **find** things, etc.
- In order to **use**, programmer doesn't need to know ***how*** the list is implemented
- Several different implementations:
  - Each has strengths / weaknesses
  - Need to understand ***inner workings*** to pick best type of list

Lists

- The idea of a list is **ABSTRACT**
- There are **concrete** implementation classes that implement list
  - E.g.
    `ArrayList, LinkedList, Array, Vector`



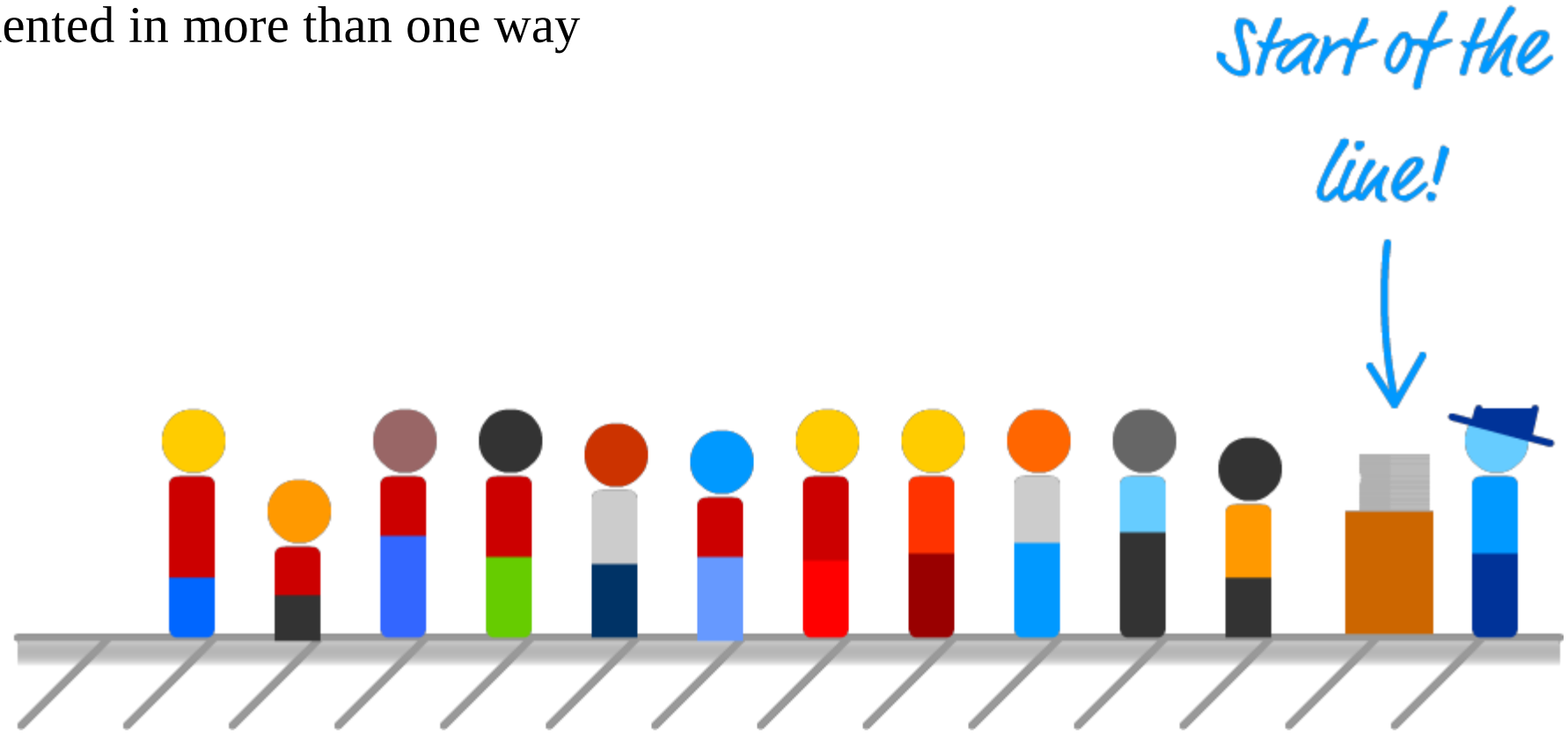
Abstract Data Types & Interfaces

What is an Abstract Data Type (ADT)?

- An **abstract data type (ADT)** is:
  - A **high-level** description of a data structure
  - A description of the methods and what they do
- Does not include:
  - Specifics about **HOW** that structure is implemented (but usually in API if you read closely)
  - Specifics about efficiency of methods
- **List** is our first ADT

Another example of an ADT.... Queue!

- A **Queue** is another example of an ADT because it could be implemented in more than one way



What is something these things all have in common?

They share a common INTERFACE

What is an interface?

Java Allows you to Define Interfaces

Real World (e.g. outlets):

- Using **standardized plugs** and **outlets** allows **reuse** of the power network by any electrical device
- A device implementing a standard North American plug is "*promising*" to support **120V AC** (what the power network supplies)

Java (Interfaces):

- Defining an **Interface** allows **reuse** of algorithms and code
- A **class** that implements an interface is "promising" (***contract***) to support methods defined in the interface
  (set of methods that any implementing class MUST include)

Java Interface

- Notice that when describing a **List**, we know some the **operations**:
  - **insert** element at end of list
  - **insert** at specific index of list
  - **get** element at specific index of list
  - **find** element in the list
- But we DO NOT describe **how** these are accomplished
  - Java allows you to describe this using **interfaces**

Example of Java Interface: LIST

```
/* Abstraction of a list that holds double values */
public interface List{

  /* Methods are NOT implemented, they are abstract */
  /* A class that is a list MUST have these methods */

  /* inserts value into this list at the end */
  public void insert(double value);

  /* inserts at the specified index */
  public void insert(double value, int index);

  /* finds the value and removes it from the list */
  public void remove(double value);

  /* finds the value and returns its index, if present */
  public int find(double value);

  /* overwrites the item at a specified index */
  public void setAt(int index, double value);

  /* returns the item at specified index */
  public double getAt(int index);
}
```

Java List API

- Note that Java already has a **List Interface**:
  - https://docs.oracle.com/javase/8/docs/api/java/util/List.html
- You can (and should) check that out on your own time!

Implementation of an actual List

- A class that IS a List
- The "**implements**" keyword means this object is a List
  - It MUST contain and implement all of the methods in the List

Implementing an interface allows a class to become more formal about the behavior it promises to provide. Interfaces form a contract between the class and the outside world, and this contract is enforced at build time by the compiler.

If your class claims to implement an interface, *all* methods defined by that interface must appear in its source code before the class will successfully compile.

Implementation of an Actual List: Vector class

```
public class Vector implements List{

  private double[] theList;

  public void insert(double value){
    /* This class will actually implement this */
    /* and actually do the inserting*/
  }

  public int find(double value){
    for(int i = 0; i < theList.length; i++){
      if(theList[i] == value) return i;
    }
    return -1; //didn't find it
  }

  /* Other list methods would be implemented below */
}
```

```
/* Abstraction of a list that holds
public interface List{

  /* Methods are NOT implemented, the
  /* A class that is a list MUST have

  /* inserts value into this list at
  public void insert(double value);

  /* inserts at the specified index
  public void insert(double value, in

  /* finds the value and removes it
  public void remove(double value);

  /* finds the value and returns its
  public int find(double value);

  /* overwrites the item at a specifi
  public void setAt(int index, double

  /* returns the item at specified in
  public double getAt(int index);
}
```

The **Vector** class the promises to include all the methods in List (methods)

The **interface**

Run-time Polymorphism

(Scary term – yet straightforward concept!)

Example of run-time polymorphism we've seen before

- Remember this? This is also an example of run-time polymorphism. Can you spot where?

```java
/* Checks for equality between two Card Objects */
@Override
public boolean equals(Object other) {
    if (!(other instanceof Card)) { // is "other" also a Card?
        return false; // "other" is not of the right data type
    }
    Card otherC = (Card)other; // Cast to Card
    return this.rank == otherC.rank && this.suit.equals(otherC.suit);
}
```

Why use Interfaces?

- What benefit do we get?
  - The primary benefit is **polymorphism**
- **Polymorphism** is a feature of object oriented languages (like Java) in which **type substitutions can be made at runtime**.
  - It is the different effects of invoking the **same method** on **different types** of objects
    - Java asks "**who are you?**" ("what is your data type?")
    - At **run-time**, Java calls the **appropriate** method
    - Could be many methods of the same name in different classes
  - Java provides this through *inheritance* and *interfaces*
- For example, if we want to write code to **sort lists**, why write a sorting method for each type of list?
  - Would be better if we had a **generic sorting method** for ALL lists
  - Then, anything that **IS a list** is sorted the exact same way.

Polymorphism Example: Object Array

- **Assume we have an Array of type Object:**

```
Object[] myArray = new Object[4];
myArray[0] = 1;                          // add int
myArray[1] = “hello";                    // add String
myArray[2] = new Object();               // add Object
myArray[3] = new Card(3, “Hearts”);      // add Card
```

What will the following print?

```
for(int i = 0; i < myArray.length; i++)
        System.out.print(myArray[i] + “ ” );
```

Polymorphism Example: Object Array

```
1 hello java.lang.Object@27c170f9 3 of Hearts
```

Each element in the array is an Object reference variable

- We call **toString()** on the Object reference
- At **run-time**, Java calls the correct toString() on the sub-class
- This is run-time polymorphism!

Polymorphism Example: Animal and Cat

```
public interface Animal {
    public void makeSound();
    public void eat();
    public void sleep();
}
public class Cat implements Animal {
    public void makeSound() {
        S.O.P("Meow!");
    } // assume eat() & sleep() exist too
}
public class Dog implements Animal {
    public void makeSound() {
        S.O.P("Woof!");
    } // assume eat() & sleep() exist too
}
```

(Assume the rest of the Cat and Dog class is implemented)

```
public class TestAnimal {
    public static void main(String
                        args[]) {
        //Create Cat and Dog
        Cat mittens = new Cat();
        Dog fido = new Dog()

        mittens.makeSound();
        fido.makeSound();
    }
)
```

OUTPUT: Meow!
Woof!

Polymorphism Example: Vector

- A **Vector** is a kind of **List** because the Vector class ***implements*** the List interface.

```
public class SortingMethods{

  public static void sort(List theList){
    /* CODE TO SORT LIST IS OMITTED */
    /* This code must perform the sort by */
    /* ONLY using methods in the interface, why? */
  }

  public static void main(String[] args){
    Vector myVector = new Vector();
    /* Add some things to the list */

    /* This call works because of polymorphism */
    /* myVector is a Vector which IS a list*/
    sort(myVector);
  }

}
```

Reminder to use: Java API

When you are stuck, the Java API is a great resource to use!

Using the Java API

- Documentation of Java classes, methods, etc.
  - VERY useful for discovering what functionality already exists in Java and how to use it.

- Some examples:
  - Object: https://docs.oracle.com/javase/8/docs/api/java/lang/Object.html
  - Scanner: https://docs.oracle.com/javase/8/docs/api/java/util/Scanner.html
  - String: https://docs.oracle.com/javase/8/docs/api/java/lang/String.html
  - ArrayList: https://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html