Interfaces and Collections in Java

Dr. Krishnendu Guha (kguha@ucc.ie)

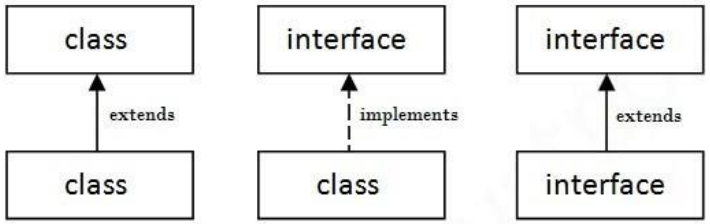
Teaching Assistants

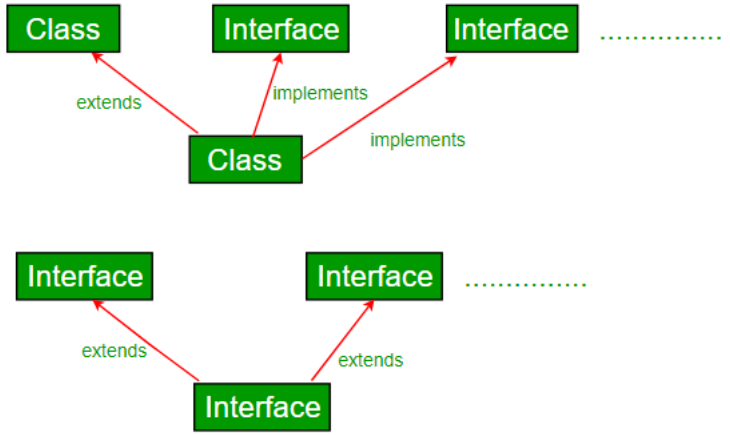
Zarrar Haider (123120583@umail.ucc.ie)

Rheena Blas (120347046@umail.ucc.ie)

Java Interfaces

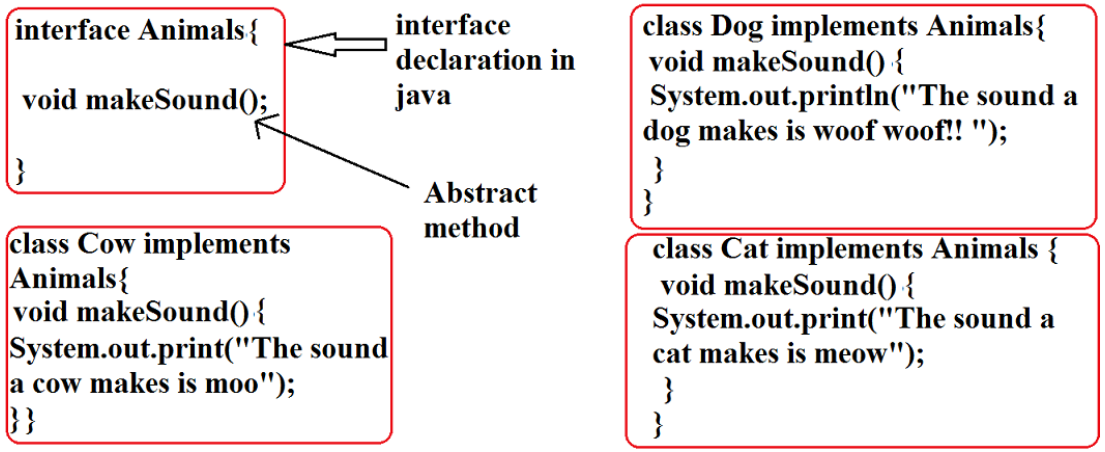
- You can think of an interface, in general, as the **set of 'services' (methods)** that *objects of a class offer to their 'clients'*
- There is a part of the Java language called an 'interface'. To distinguish, we will call it a Java interface.
- A Java interface is just **a collection of abstract methods** (i.e. we **give the signatures but not the bodies**).
- **Advantage**: Using Java interfaces polymorphically *gives you code that is more easily modified*





- A Java interface is a **collection of abstract methods** - Not a class!
- Difference from a class: A class **implements an interface**, thereby **inheriting the abstract methods of the interface**
- It **expands on the abstract method concept** *because a class can implement multiple interfaces* whereas it can *only inherit from a single super class*
- Interfaces have **no constructors!**
- Writing an interface is similar to writing a class, but they are two different concepts

```
// interface
interface Animal {
        public void animalSound(); // interface method (does not have a body)
        public void run(); // interface method (does not have a body)
}
```

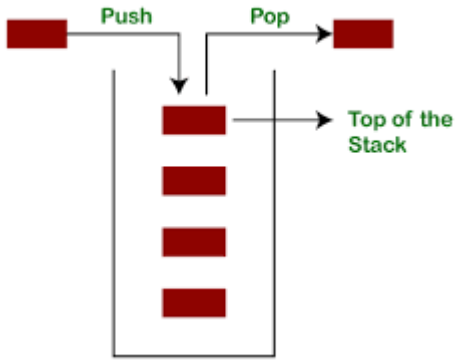


Class Definitions vs Java Interfaces

- Contrast:
  - a **class definition** can *contain constructors, variables, and methods* (comprising each method's *signature and body*, unless abstract)
  - a **Java interface** can *contain only public method signatures and (advanced point) named constants*
- A Java interface **acts like a specification**
  - it specifies, e.g., *what it means to be a list*
  - i.e. *to be a stack*, an object must offer (at least) these methods

Stack as Java Interface

```
interface Stack {
        public void push(Object element);
        public Object pop();
        public Object top();
        public int length();
        public boolean isEmpty();
}
```



Implementing a Java Interface

When you **write a class definition, you can declare that it 'implements a Java interface'**, i.e. it meets the specification:

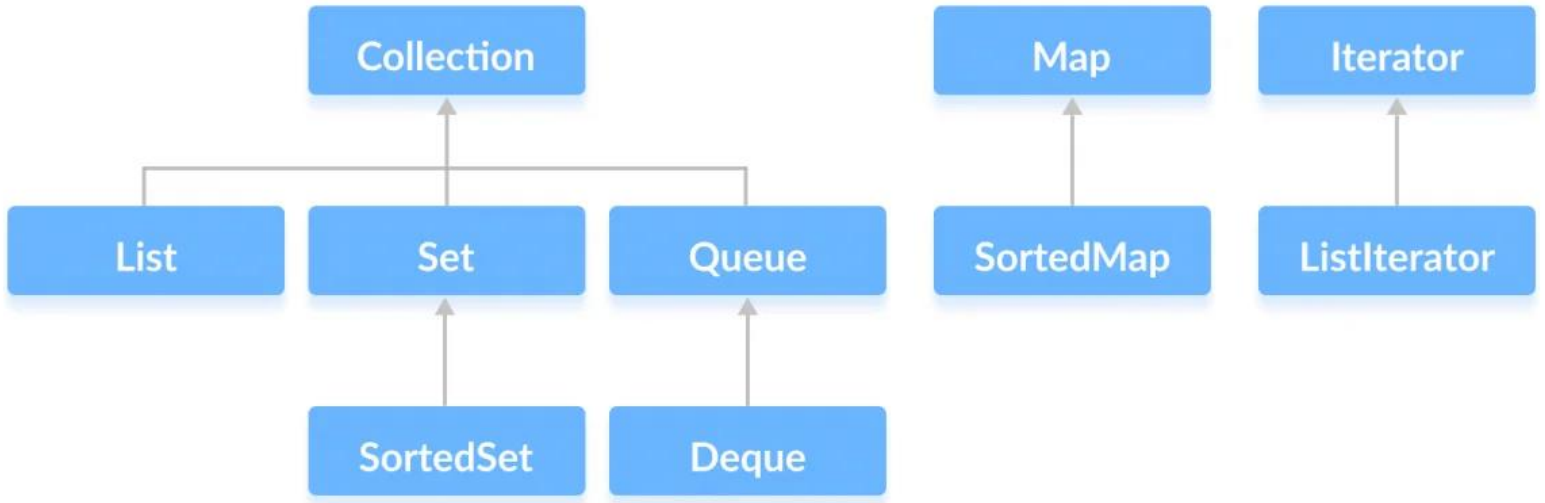
```
class ArrayBasedStack implements Stack {
        private Object[] s;
        private int capacity;
        private int size;

        public ArrayBasedStack(int capacity) {
                this.capacity = capacity;
                s = new Object[capacity];
        }

        public void push(Object element) {
                if (size == capacity) {
                        return;
                }
                s[size] = element;
                size = size + 1;
        }
```

```
public Object pop() {
        if (size == 0) {
                return null;
        }
        size = size - 1;
        return s[size];
}
public Object top() {
        if (size == 0) {
                return null;
        }
        return s[size - 1];
}
public int length() { return size; }
public boolean isEmpty() { return size == 0; }
}
```

Collections

- Java **provides a library of classes called the Collection classes** that *can be used as data structures* **that *contain instances of objects***
- These **classes store references to instances of any class**
- They can be **used to *store instances*** *of String, Integer, Double, Float, Character, Boolean and any user defined class*
- However, they **cannot be used to store variables** of type int, char, boolean, double or float
- All classes stored should be collection compliant. That is, they must implement the following:
  - equals()
  - compareTo()
  - hashCode()
  - toString()
- Library supports *three related categories of collection: Sets, Lists, Queues*

Java Collections Framework



Lists

- A list is a **collection of items ordered by their positions** in the list
- Possible to **reference items** by *referencing index value*

- **Two lists equal** if they *contain the same elements in the same order*
- Examples: ArrayList and LinkedList

List Interface

- **Standard Java arrays** are of a **fixed length**
- After arrays are created, **they cannot grow or shrink**, which means that **you must know in**
- **advance how many elements an array will hold**

- The **ArrayList class extends AbstractList** and **implements the List interface**
- ArrayList **supports dynamic arrays** that can **grow as needed**
- Array lists are **created with an initial size**
- When this **size is exceeded, the List is automatically enlarged**
- When **objects are removed, the array may be shrunk**

- // Constructors:

  ArrayList<E>()

  ArrayList<E>(Collection)

  LinkedList<E>()

  LinkedList<E>(Collection)

// Some methods:

  add(E x)

  remove(E x)

  boolean contains(E x)

  int size()

  toString()

  boolean isEmpty()

  clear()

  E get(int index);

  add(int index, E x);

  E set(int index, E x);

  E remove(int index)

  int indexOf(E x);

  E - These are generic data structures (we will see these later)

Example: array list of strings

```
public class ArrayTest1 {
	public static void main(String args[]){
		ArrayList<String> lst = new ArrayList<String>();
		lst.add("Joe");
		lst.add("Cat");
		lst.add("House");
		System.out.println(lst.toString());
		lst.add(1,"Pat");
		System.out.println(lst.toString());
		lst.set(3,"Dog");
		System.out.println(lst);
		lst.remove("Joe"); //remove element
		System.out.println(lst);
		//print data using iterator
		Iterator<String> t = lst.iterator();
		while(t.hasNext()) {
			String s = t.next();
			System.out.print(s + " ");
		}
}
```

```
import java.util.*;
public class ArrayTest1 {
	public static void main(String args[]){
		ArrayList<String> lst = new ArrayList<String>();
		lst.add("Joe");lst.add("Cat");
		lst.add("House");
		System.out.println(lst);
		lst.add(1,"Pat");
		System.out.println(lst);
		lst.set(3,"Dog");
		System.out.println(lst);
		lst.remove("Joe"); //remove element
		System.out.println(lst);
		//print data using for-each loop
		for(String s : lst) {
			System.out.print(s + " ");
			System.out.println();
		}
}
```


Thank you