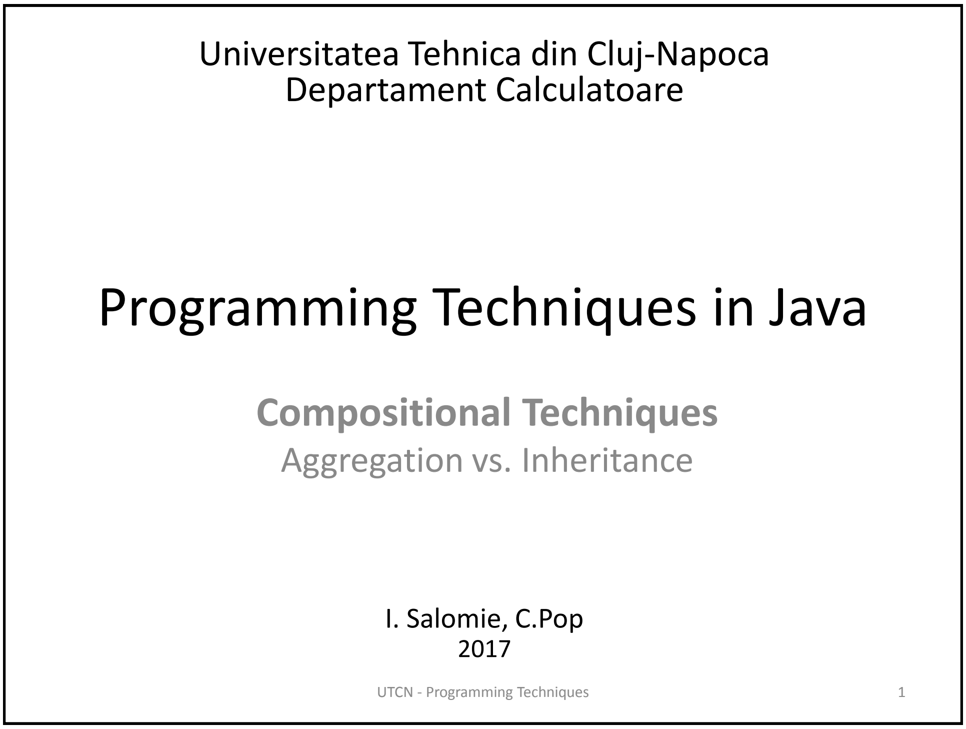


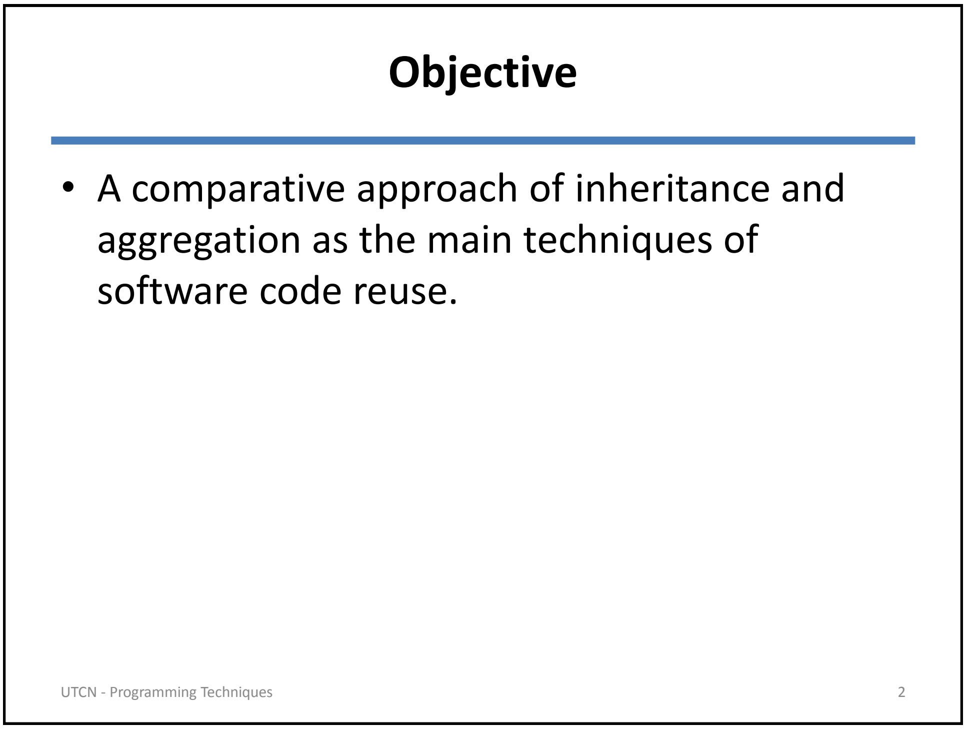
Universitatea Tehnica din Cluj-Napoca
Departament Calculatoare

Programming Techniques in Java

Compositional Techniques

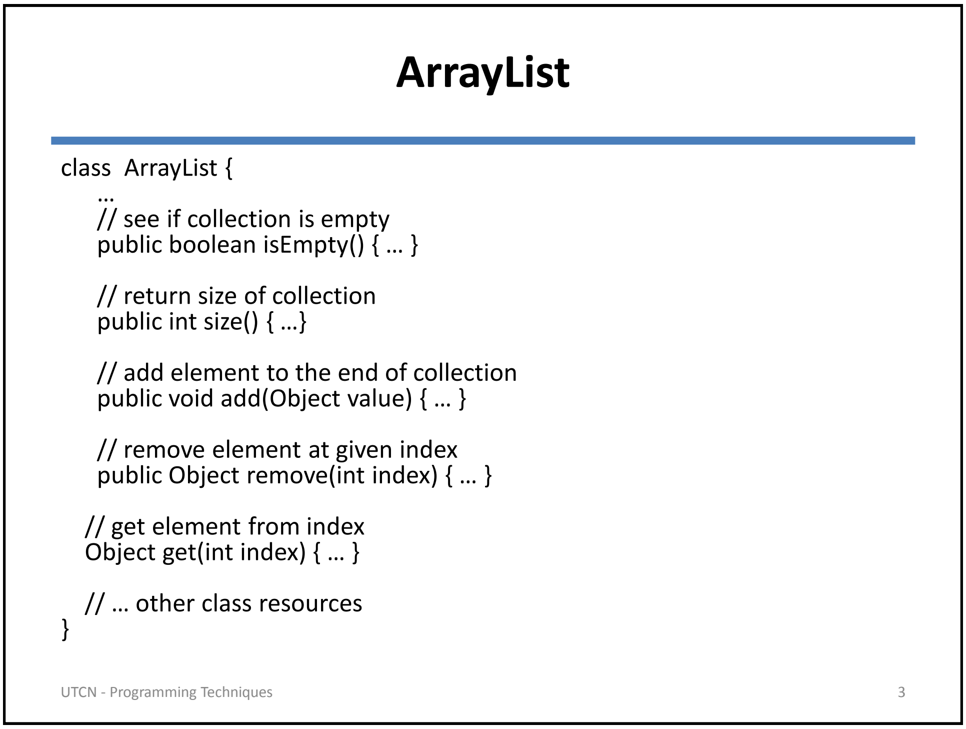
Aggregation vs. Inheritance

I. Salomie, C.Pop
2017



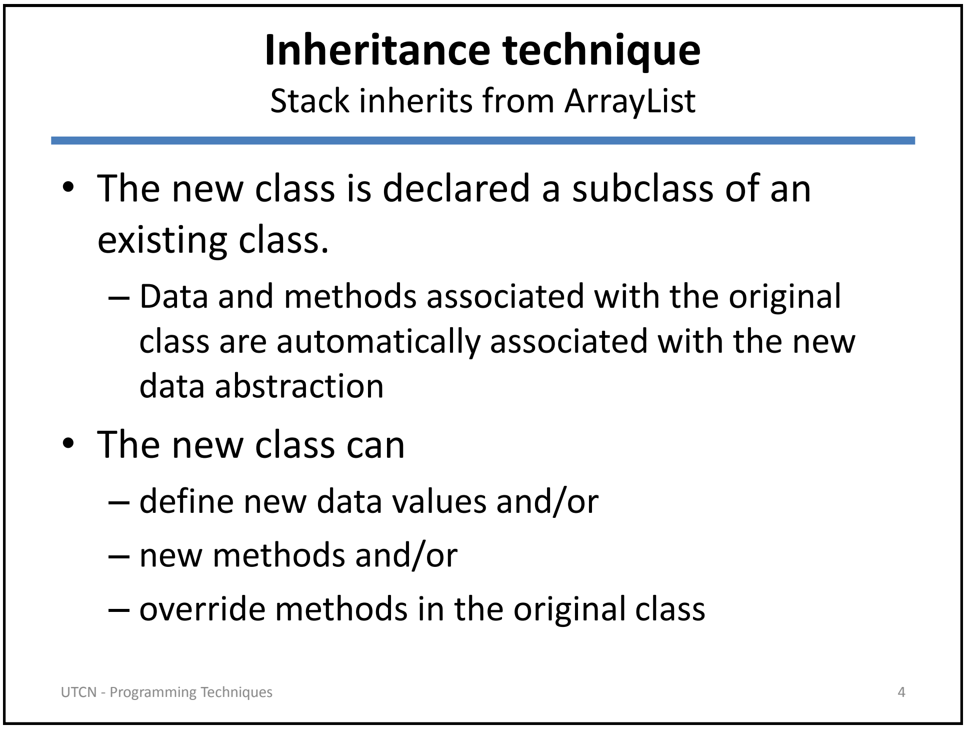
Objective

- A comparative approach of inheritance and aggregation as the main techniques of software code reuse.



ArrayList

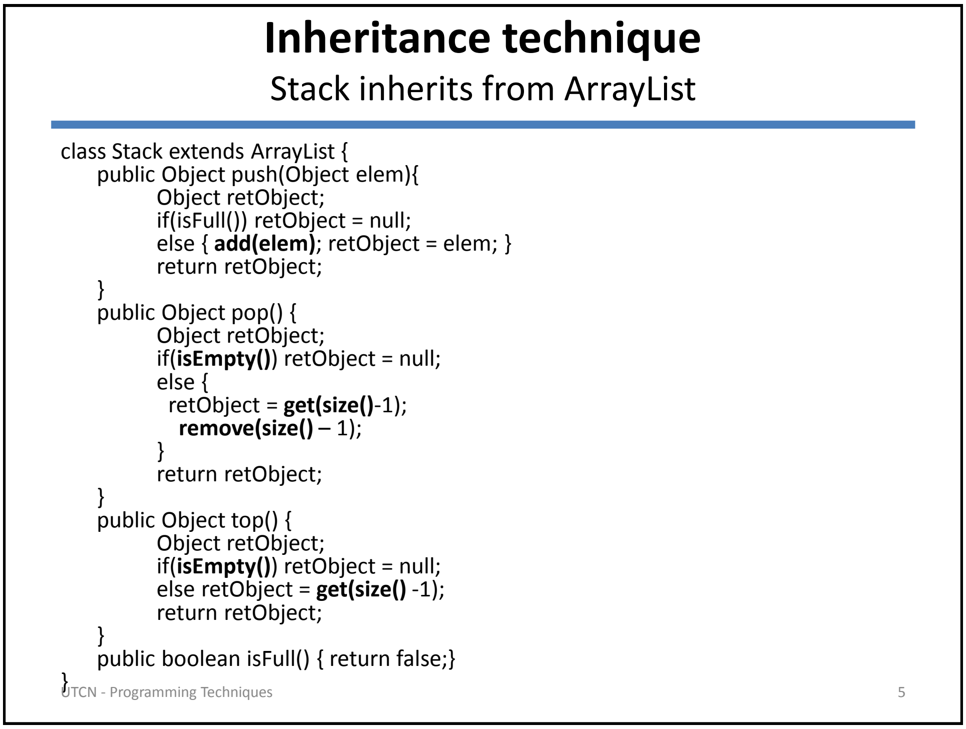
```
class  ArrayList {
    …
    // see if collection is empty
    public boolean isEmpty() { … }

    // return size of collection
    public int size() { …}

    // add element to the end of collection
    public void add(Object value) { … }

    // remove element at given index
    public Object remove(int index) { … }

   // get element from index
   Object get(int index) { … }

   // … other class resources
}
```



Inheritance technique

Stack inherits from ArrayList

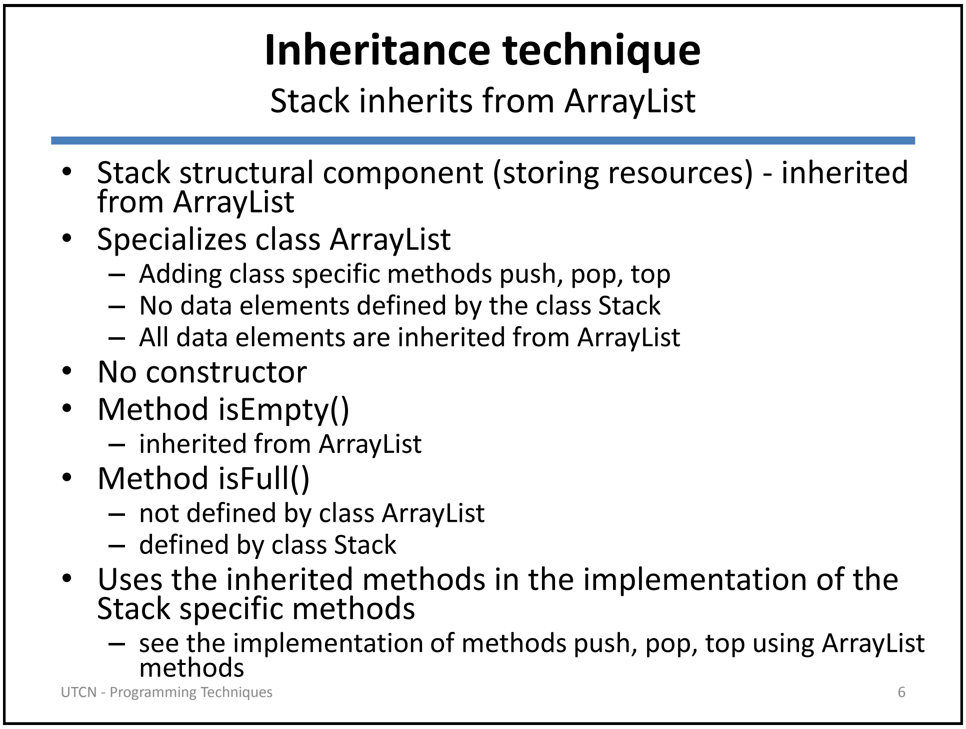
- The new class is declared a subclass of an existing class.
  - Data and methods associated with the original class are automatically associated with the new data abstraction
- The new class can
  - define new data values and/or
  - new methods and/or
  - override methods in the original class



Inheritance technique

Stack inherits from ArrayList

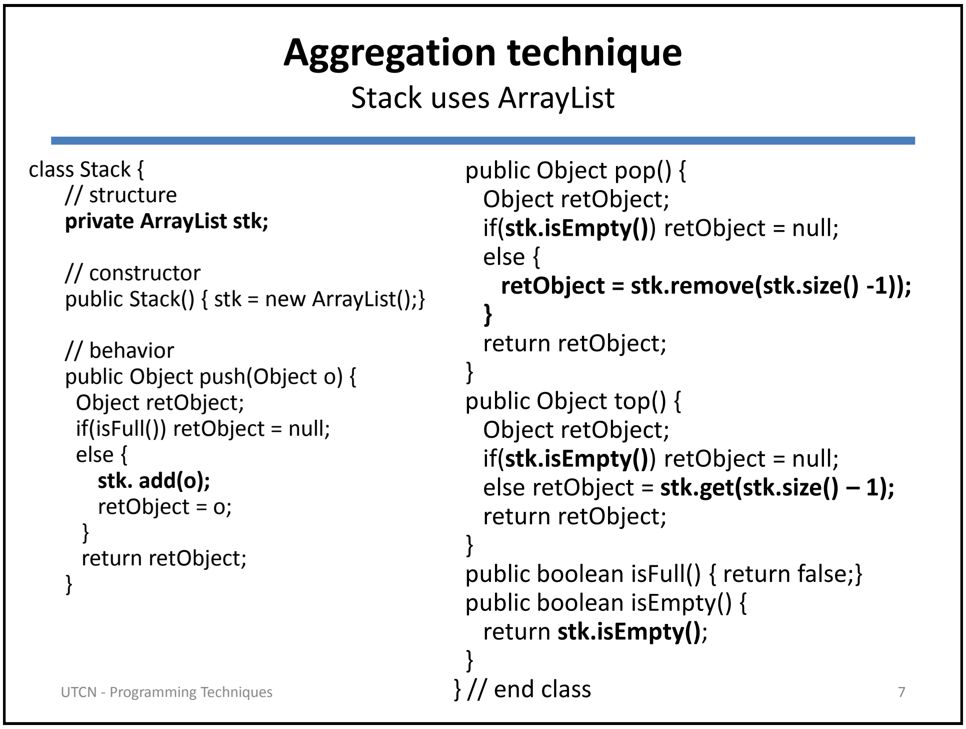
```
class Stack extends ArrayList {
     public Object push(Object elem){
               Object retObject;
               if(isFull()) retObject = null;
               else { add(elem); retObject = elem; }
               return retObject;
     }
     public Object pop() {
               Object retObject;
               if(isEmpty()) retObject = null;
               else {
                 retObject = get(size()-1);
                   remove(size() – 1);
               }
               return retObject;
     }
     public Object top() {
               Object retObject;
               if(isEmpty()) retObject = null;
               else retObject = get(size() -1);
               return retObject;
     }
     public boolean isFull() { return false;}
}
```



Inheritance technique

Stack inherits from ArrayList

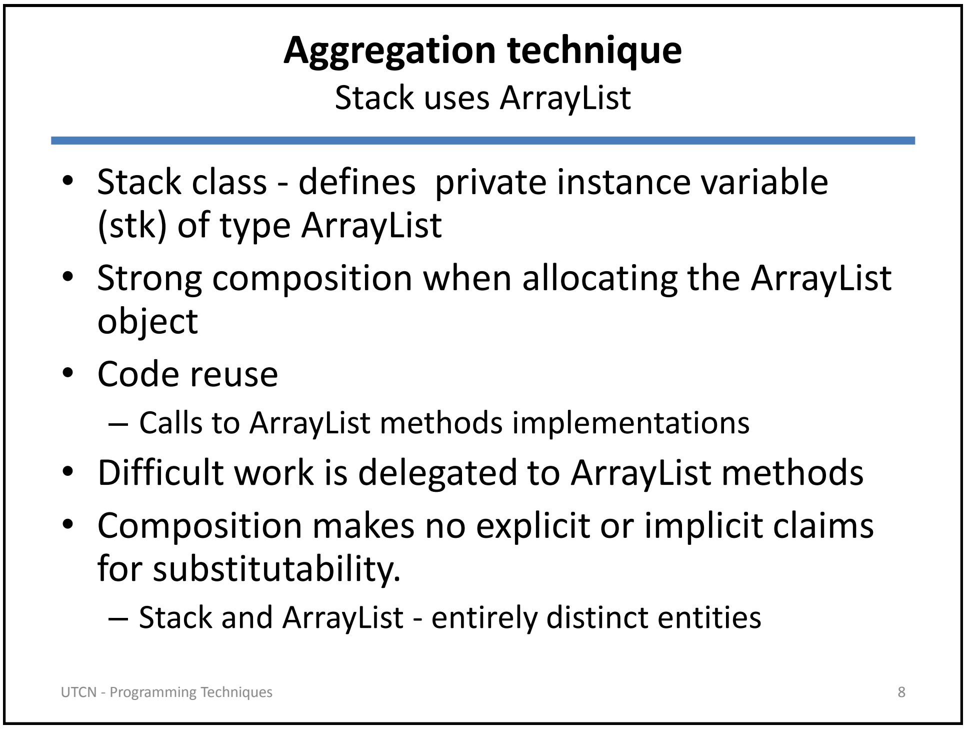
- Stack structural component (storing resources) - inherited from ArrayList
- Specializes class ArrayList
  - Adding class specific methods push, pop, top
  - No data elements defined by the class Stack
  - All data elements are inherited from ArrayList
- No constructor
- Method isEmpty()
  - inherited from ArrayList
- Method isFull()
  - not defined by class ArrayList
  - defined by class Stack
- Uses the inherited methods in the implementation of the Stack specific methods
  - see the implementation of methods push, pop, top using ArrayList methods



Aggregation technique

Stack uses ArrayList

```
class Stack {
    // structure
    private ArrayList stk;

    // constructor
    public Stack() { stk = new ArrayList();}

    // behavior
    public Object push(Object o) {
      Object retObject;
      if(isFull()) retObject = null;
      else {
         stk. add(o);
         retObject = o;
       }
       return retObject;
    }
public Object pop() {
   Object retObject;
   if(stk.isEmpty()) retObject = null;
   else {
     retObject = stk.remove(stk.size() -1));
   }
   return retObject;
}
public Object top() {
   Object retObject;
   if(stk.isEmpty()) retObject = null;
   else retObject = stk.get(stk.size() – 1);
   return retObject;
}
public boolean isFull() { return false;}
public boolean isEmpty() {
   return stk.isEmpty();
}
} // end class
```



Aggregation technique

Stack uses ArrayList

- Stack class - defines private instance variable (stk) of type ArrayList
- Strong composition when allocating the ArrayList object
- Code reuse
  - Calls to ArrayList methods implementations
- Difficult work is delegated to ArrayList methods
- Composition makes no explicit or implicit claims for substitutability.
  - Stack and ArrayList - entirely distinct entities

Comparison of the two techniques

- Aggregation
  - Simple technique
  - All the involved elements are clearly highlighted
  - Clearly shows all available operations for the abstraction Stack
  - Longer code, clear operations
  - A user (programmer) looks only at the class supplied code in order use stack facilities
  - Compositions are very simple to be changed by using other structural components.
    - Examples
  - Better separates the two abstractions

Comparison of the two techniques

- Inheritance
  - A user (programmer) should study and understand the methods of the superclass
  - Less code
  - Operations are more difficult to understand
  - Some methods are already implemented
    - could be directly reused
    - Example: isEmpty
  - Semantic differences
  - Less overhead in execution, than the composition

Comparison of the two techniques

- Inheritance (cont.)
  - Inappropriate using methods of the superclass
    - Examples
  - Error prone
    - Examples
  - Allows using the new abstraction as an argument in an existing *polymorphic* method.
  - Better execution time

Comparison of the two techniques

- The advantage of composition over inheritance
  - the delay in binding time
- Inheritance
  - The link between child class and parent class
    - Established during compile time
    - Immutable (cannot be later modified)
- Composition
  - Dynamic composition
  - The link between the new abstraction and the older abstraction is created/changed at run time