E0-245: ASP

Lecture 8: OOPs Templates and Generics

Dipanjan Gope

Questions from Lecture 6

- Does a class in C++ require “virtual” to be polymorphed?

Module 1: OOPs and DS

JAVA and C++ mainly: (others C# and Objective-C)

- Object oriented programing: Classes and objects
- Inheritance, polymorphism, abstract class, const
- **Templates and generics**
- Data structures
- Standard library, JCF, STL
- Complexity analysis
- Multithreading and synchronization
- Good programming styles

The Basic Idea

```
int square (int x)
{
    return x*x;
}

float square (float x)
{
    return x*x;
}

double square (double x)
{
    return x*x;
}
```

```
template <class T>
T square(T x)
{
    return x*x;
};
```

Class Array of Integers: Example code

```
public class Array
{
    private int[] data;
    private int size;
    private int pos;

    public Array(int numElements)
    {
        size = numElements;
        pos = 0;
        data = new int[size];
    }
}
```

Class Array of Integers: Generic code

```
public class Array<T>
{
    private T[] data;
    private int size;
    private int pos;

    public Array(int numElements)
    {
        size = numElements;
        pos = 0;
        data = (T[]) new Object[size];
    }
}
```

https://www.youtube.com/watch?v=5SfFVd1FW64

Generics for collection framework

```
List list = new ArrayList();
list.add("hello");
Integer s = (Integer) list.get(0);
```

```
List<String> list = new ArrayList<String>();
list.add("hello");
Integer s = (Integer) list.get(0);
```

Runtime Error to Compile Error!

Generics for collection framework

```
List list = new ArrayList();
list.add("hello");
String s = (String) list.get(0);
```

```
List<String> list = new ArrayList<String>();
list.add("hello");
String s = list.get(0);
```

No Type Casting

Generics

- Introduced in J2SE 5.0, 2004
- Reuse of code
- Type checks at compile time
- No need for type casts

Generic

- Class

```
List<String> list = new List<String>();
```

- Method

```
public static <T> void swap(T[] a, int i, int j)
{  T temp = a[i];  a[i] = a[j];  a[j] = temp; }
```

- Interface

```
public interface accessList<T>
{ void add(T newElement);
  T get(int n); }
```

Bounded Parameters

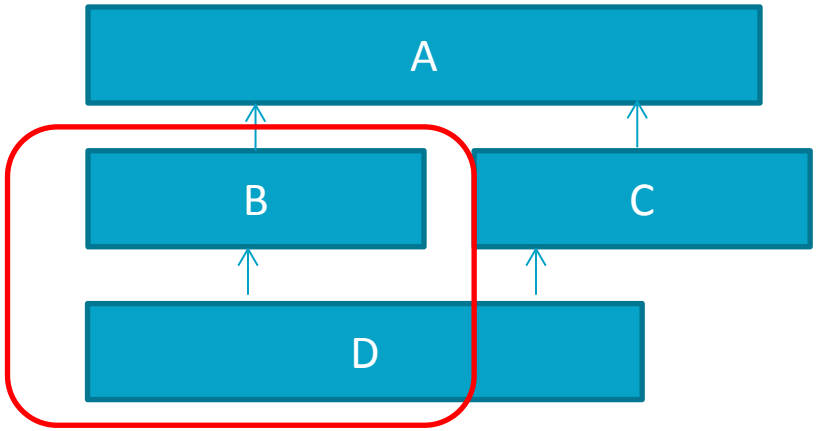
```
public static <T> T max(T x, T y)
{ return x > y ? x : y; }
```

```
public static <T extends Comparable> T max(T x, T y)
{ return x.compareTo(y) > 0 ? x : y; }
```

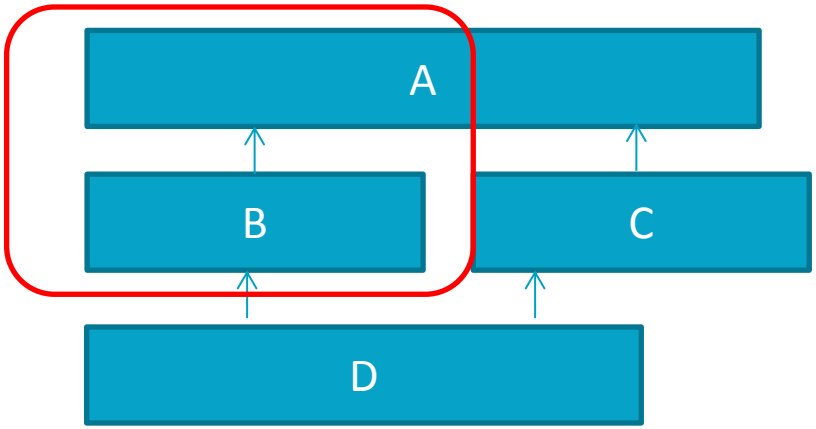
```
public interface Comparable<T> { public int compareTo(T o); }
```

Wild Cards

- public static void process(List<? extends B> list) {}



- public static void process(List<? super B> list) {}



Type Erasure for Generics

```
public class Node<T> {

    private T data;
    private Node<T> next;

    public Node(T data, Node<T> next) }
        this.data = data;
        this.next = next;
    }

    public T getData() { return data; }
    // ...
}
```

```
public class Node {

    private Object data;
    private Node next;

    public Node(Object data, Node next) {
        this.data = data;
        this.next = next;
    }

    public Object getData() { return data; }
    // ...
}
```

http://docs.oracle.com/javase/tutorial/java/generics/genTypes.html

Type Erasure for Generics

```
public class Node<T extends Comparable<T>> {

    private T data;
    private Node<T> next;

    public Node(T data, Node<T> next) {
        this.data = data;
        this.next = next;
    }

    public T getData() { return data; }
    // ...
}
```

```
public class Node {

    private Comparable data;
    private Node next;

    public Node(Comparable data, Node next) {
        this.data = data;
        this.next = next;
    }

    public Comparable getData() { return data; }
    // ...
}
```

http://docs.oracle.com/javase/tutorial/java/generics/genTypes.html

Templates in C++

- Method

```
template<class T>
T add(T n1, T n2)
{ T result; result = n1 + n2; return result; }
```

- Class

```
template <class T>
class Array
{
 protected:
      T* arrayElements;
 }
```

C++ Templates vs JAVA Generics

- Compiler:

  - Filling template parameters vs. Type erasure

- Type parameters:

  - primitives vs. no primitives

  - bounding of arguments: no vs. yes