



CSE 215: Programming Language II Lab

Lab – 14 Interface and Generics

Objective:

- To understand interface and its usage
- To utilize interface to ensure reusability of existing code
- To detect errors at compile time
- Ensure reusability of code

Interface

An interface is a structure which specifies the **common behavior** of objects of different classes. It contains only the constants and the abstract methods just like an abstract class. All data fields are **public final static** and all methods are **public abstract** in an interface. So if you miss to mention these, it will still work.

```
public class GeometricObject implements Resizable
```

```
public interface Resizable {  
  
    public static final double MAX= 100;  
    public abstract void resize (double percentage);  
}
```

Equivalent

```
public interface Resizable {  
  
    double MAX = 100;  
    void resize (double percentage);  
}
```

```
public class Circle extends GeometricObject  
implements Resizable{  
    .....  
    .....
```

```
public void resize(double percentage){  
    if(percentage<=MAX){  
        radius += radius*percentage/100;  
    }  
}
```

The Comparable Interface

```
public interface Comparable{    int  
compareTo(GeometricObject o);  
}
```

```
public class GeometricObject implements  
Comparable <GeometricObject> {  
    .....  
    ....  
}
```

<pre> public int compareTo(GeometricObject o){ if (getArea() > (o.getArea())) return 1; elseif (getArea()<(o.getArea())) return -1; else return 0; } </pre>	<pre> GeometricObject c1 = new Circle(20); Circle c2 = new Circle(20); System.out.println(c2.compareTo(c)); // 0 </pre>
---	--

Multiple Interfaces

```

public class GeometricObject implements Comparable, Resizable, Movable, ... {
}

```

Generics

Generics enable you to detect errors at compile time rather than at runtime

<pre> package java.lang; public interface Comparable { public int compareTo(Object o) } </pre>	<pre> package java.lang; public interface Comparable<T> { public int compareTo(T o) } </pre>
--	--

Here, **<T>** represents a formal generic type, which can be replaced later with an actual concrete type.

Replacing a generic type is called a generic instantiation.

By convention, a single capital letter such as **E** or **T** is used to denote a formal generic type.

<pre> Comparable<Date> c = new Date(); System.out.println(c.compareTo("red")); </pre>

c is a reference variable whose type is **Comparable<Date>** and invokes the **compareTo** method to compare a **Date** object with a string. This code generates a compile error.

Define Generic class and Interfaces

GenericStack<E>	
<pre> -list: java.util.ArrayList<E> +GenericStack() +getSize(): int +peek(): E +pop(): E +push(o: E): void +isEmpty(): boolean </pre>	<pre> An array list to store elements. Creates an empty stack. Returns the number of elements in this stack. Returns the top element in this stack. Returns and removes the top element in this stack. Adds a new element to the top of this stack. Returns true if the stack is empty </pre>

GenericStack.java

```
public class GenericStack<E> {
    private java.util.ArrayList<E> list = new java.util.ArrayList<>();

    public int getSize() {
        return list.size();
    }

    public E peek() {
        return list.get(getSize() - 1);
    }

    public void push(E o) {
        list.add(o);
    }

    public E pop() {
        E o = list.get(getSize() - 1);
        list.remove(getSize() - 1);
        return o;
    }

    public boolean isEmpty() {
        return list.isEmpty();
    }

    @Override
    public String toString() {
        return "stack: " + list.toString();
    }
}
```

Main Method:

```
GenericStack<String> stack1 = new GenericStack<>();
stack1.push("London");
stack1.push("Paris");
stack1.push("Berlin");
GenericStack<Integer> stack2 = new GenericStack<>();
stack2.push(1); // autoboxing 1 to new Integer(1)
stack2.push(2);
stack2.push(3);
```

Generic Method

To declare a generic method, you place the generic type <E> immediately after the **keyword static** in the method header. For example,

```
public static <E> void print(E[] list)
```

To invoke a generic method, prefix the method name with the actual type in angle brackets. For example,

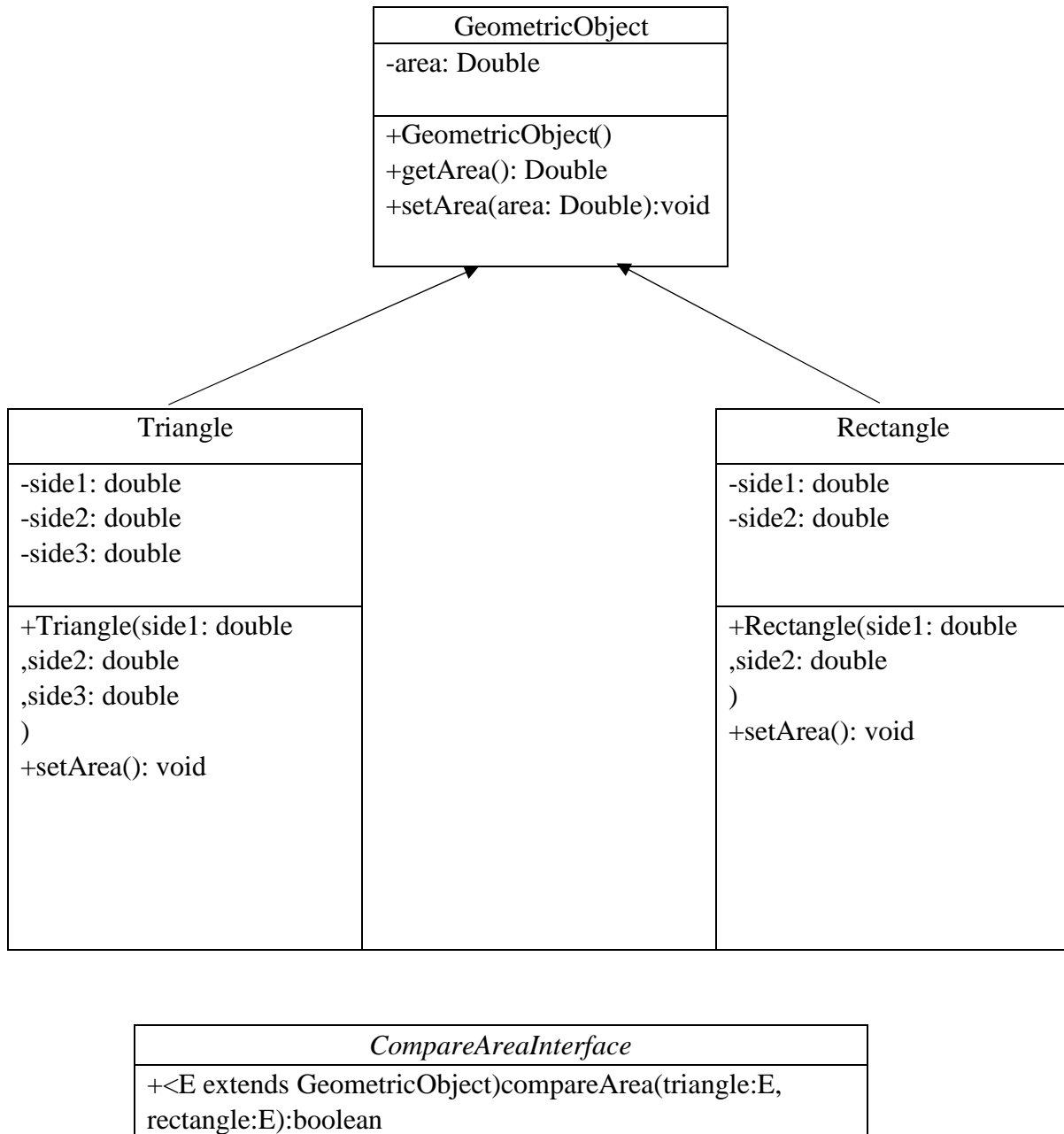
```
GenericMethodDemo.<Integer>print(integers);  
GenericMethodDemo.<String>print(strings);
```

A generic type can be specified as a subtype of another type. Such a generic type is called **bounded**.

The bounded generic type **<E extends Object>** specifies that E is a generic subtype of Object.

```
public class BoundedTypeDemo {  
    public static void main(String[] args) {  
        Rectangle rectangle = new Rectangle(2, 2);  
        Circle circle = new Circle(2);  
  
        System.out.println("Same area? " +  
            equalArea(rectangle, circle));  
    }  
  
    public static <E extends GeometricObject> boolean equalArea(  
        E object1, E object2) {  
        return object1.getArea() == object2.getArea();  
    }  
}
```

Lab Task



Write a test class that implements *CompareAreaInterface* interface; in the main method create two objects of a Triangle, and a rectangle. With the help of *compareArea()*, compare area of two objects and print appropriate message.