Java Generics – Wildcards

By: Anupam Chanda

Generics and Subtyping

We start to run into some new issues when we do some things that seem "normal". For instance, the following seems reasonable:

```
Box<Number> numBox = new Box<Integer>(31);
```

Compiler comes back with an "Incompatible Type" error message.

This is because `numBox` can hold only a `Number` object and nothing else, not even an object of type `Integer` which is a subclass of `Number`.

`Box<T>` is not a subclass of `Box<E>` even if `T` is a subclass of `E`.

```
//Consider the following lines of code
Box<String> strBox = new Box<String>("Hi");//1
Box<Object> objBox = strBox;//2 - compilation error
objBox.setData(new Object());//3
String s = strBox.getData();//4 - an Object to a String!
```

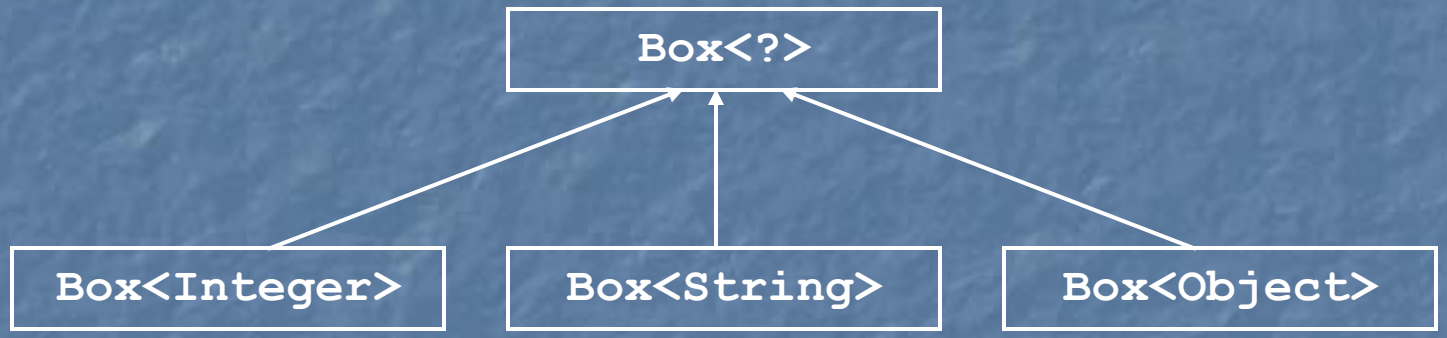
Unbounded Wildcards

We want to write a method to print any `Box`.

```
public static void printBox(Box<Object> b) {
        System.out.println(b.getData());
}

Box<String> strBox = new Box<String>("Hi");
printBox(strBox);//compilation error


public static <T> void printBox(Box<T> b) {
        System.out.println(b.getData());
}//parameterized method

public static void printBox(Box<?> b) {
        System.out.println(b.getData());
}//using unbounded wildcard
```

Unbounded Wildcards (Contd.)

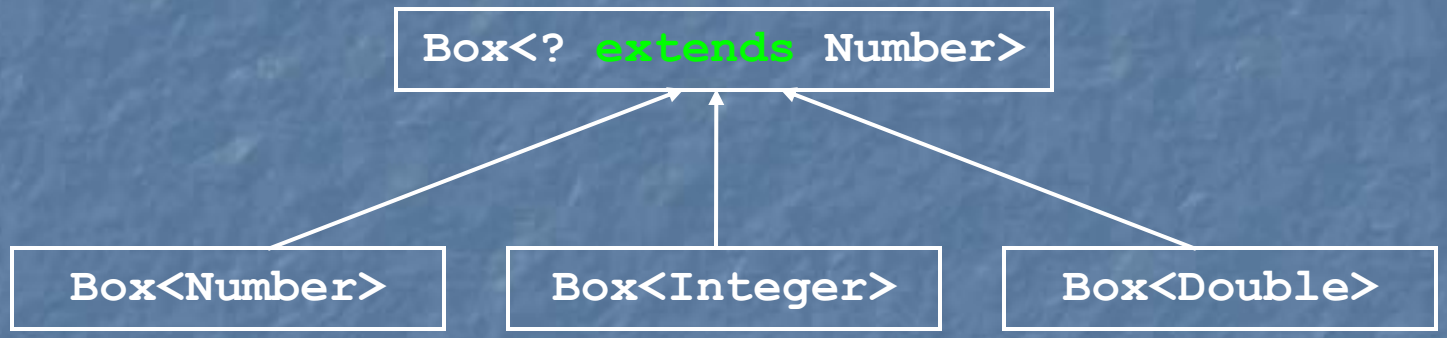
`Box<?>` is a superclass of `Box<T>` for any `T`.



Unbounded wildcards are useful when writing code that is completely independent of the parameterized type.

Upper Bounded Wildcards

"A `Box` of any type which is a subtype of `Number`".




```
Box<? extends Number> numBox = new Box<Integer>(31);
```

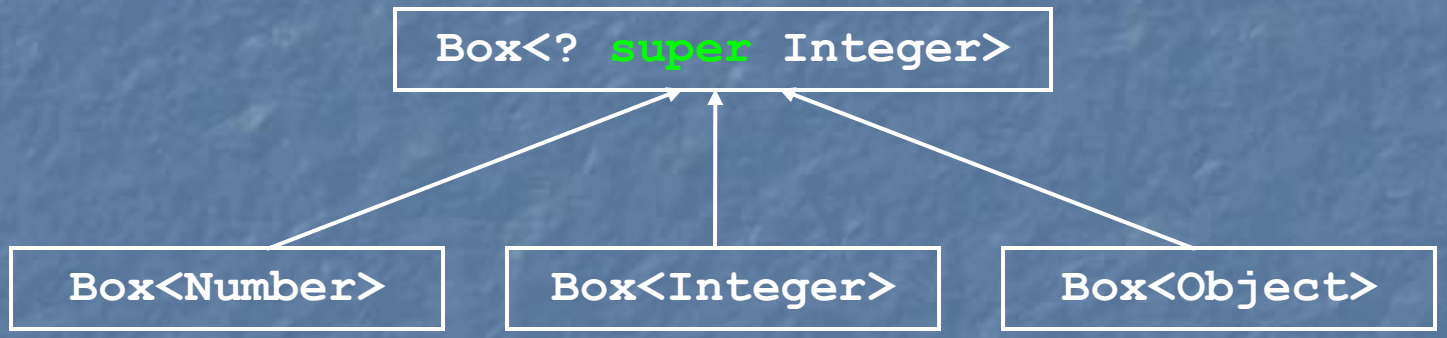
`<? extends E>` is called "*upper bounded wildcard*" because it defines a type that is bounded by the superclass `E`.

Upper Bounded Wildcards (Contd.)

```
public class Box<E> {
  public void copyFrom(Box<E> b) {
    this.data = b.getData();
  }
}
//We have seen this earlier
//We can rewrite copyFrom() so that it can take a box
//that contains data that is a subclass of E and
//store it to a Box<E> object

public class Box<E> {
  public void copyFrom(Box<? extends E> b) {
    this.data = b.getData();//b.getData() is a
                            //subclass of this.data
  }
}

Box<Integer> intBox = new Box<Integer>(31);
Box<Number> numBox = new Box<Number>();
numBox.copyFrom(intBox);
```

Lower Bounded Wildcards

"A `Box` of any type which is a supertype of `Integer`".



`<? super E>` is called a "*lower bounded wildcard*" because it defines a type that is bounded by the subclass `E`.

Lower Bounded Wildcards (Contd.)

Suppose we want to write `copyTo()` that copies data in the opposite direction of `copyFrom()`.
`copyTo()` copies data from the host object to the given object.

This can be done as:

```
public void copyTo(Box<E> b) {
  b.data = this.getData();
}
```

Above code is fine as long as `b` and the host are boxes of exactly same type. But `b` could be a box of an object that is a superclass of `E`.

This can be expressed as:

```
public void copyTo(Box<? super E> b) {
  b.data = this.getData();
  //b.data() is a superclass of this.data()
}
```

```
Box<Integer> intBox = new Box<Integer>(31);
Box<Number> numBox = new Box<Number>();
intBox.copyTo(numBox);
```