Java Generics – Bounded Types

Lab setup

- Use the same project from the previous lab on generics.
- We will be writing mostly in the main class

Bounded type parameters

Sometimes you may want to restrict the types which can be used as type arguments in a parameterized type. This can be done using the keyword `extends` after the type parameter in the type parameter list.

```
public static <T extends Comparable<T>> int doSomething(T[] arr, T elem){
    ///...
}
```

Here, `doSomething` is restricted to types which are `Comparable`.

Graded Submission Task 1

1. Implement `doSomething` to count the number of elements in `arr` which are less than or equal to `elem`
2. Every `Comparable` implements `compareTo` which you can use to determine order.
3. Show that your implementation works in the driver.

** STOP **

Ensure you have saved the source files in your project. Make a copy of your Eclipse project at this point (you can do this from your system's file explorer). Name the file labelling it as Task 1. You will submit this copy at the end of this lab.

Wildcards

4. Suppose we wanted to write a generic method to print the contents of a `List`. We might try

   ```
   public static void printList(List<Object> list)
   ```

   However, this would not work because not all lists are subclasses of `List<Object>`. Instead, we can use a wildcard to say the functionality of this method does not depend on the type parameter. The wildcard character in this case is the question mark.

```
public static void printList(List<?> list)
```

5. We can also bound the types of the unknown type using keywords `super` or `extends`. Using `super` specifies a lower bound and `extends` specifies an upper bound. We can only specify one, and not both.

   Writing `<? super Integer>` bounds the type to be `Integer` or one of its superclasses.

   Writing `<? extends IOException>` bounds the type to be `IOException` or one of its subclasses.

Graded Submission Task 2

Implement generic methods for Π (product) and Σ (sum) on lists using a bounded wildcard for Numbers. Show that your implementation works in the driver.

Applying Π to a `List` should return the product of all its elements.

$$\prod_i^N x_i = x_0 \cdot x_1 \cdot \ldots \cdot x_N$$

Applying Σ to a `List` should return the sum of all its elements.

$$\sum_i^N x_i = x_0 + x_1 + \ldots + x_N$$

For these methods, the return type should be BigDecimal.

** STOP **

Ensure you have saved the source files in your project. Make a copy of your Eclipse project at this point (you can do this from your system's file explorer). Name the file labelling it as Task 2. You will submit this copy at the end of this lab.

"In" and "Out" variables

"In" variables can be described as arguments which provide data to be operated on to a method. "Out" variables can be described as the arguments which will hold the data for use elsewhere from a method. E.g., In the method `copy(src, dest)` `src` is the "in" variable, and `dest` is the "out" variable. The following are guidelines for wildcard use laid out by Oracle.

- "in" variables are defined with an upper bounded wildcard.
- "out" variables are defined with a lowerbounded wildcard.
- When "in" varables can be accessed using methods defined in Object, use an unbounded wildcard.
- When code needs to access a variable as both an "in" and "out" variable, do not use a wildcard.

Wildcards should not be used as return types as it will force others who use your code to deal with the wildcard.

Lab wrap-up

In this lab we continue the discussion on generics with bounded types and wildcards.

Type parameters can be bounded using the keyword extends after the type parameter. The example we used is shown in Figure 1. The types that can be used with this method must be those which implement Comparable.

If we try to define a print method for all lists generically, Java will most likely compile the method for types List<Object> because of type erasure. Not all lists are subclasses of List<Object> though. Instead, we can use a wildcard in place of the type parameter to say it does not matter what type the elements of the list are.

Submission notes for graded tasks:

- Submit a zipped folder containing the Eclipse projects for each graded task.
- For each project, ensure you have included the src folder containing the .java file, the .classpath file, and the .project file.
- Ensure that each project is named appropriately to correspond with each task.
- Zip the projects together and name the zipped folder "Lab18_cslogin", where cslogin is your login ID for the computers at the Department of Computer Science at ODU.
- Submit the zipped file to the respective Canvas link.

Rubrics	
Task 1 (points allotted) - Compilation (2) - doSomething implemented (4) - Driver demonstrates implementation is correct (4)	Task 2 (points allotted) - Compilation (2) - Generic method signatures are correct (2) - Sum is implemented (2) - Product is implemented (2) - Driver demonstrates implementation is correct (2)