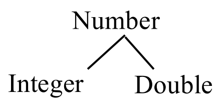
Upper-bounded type parameters

We give examples of generic upper-bounded type parameters.



Function sum

Function sum returns the sum of the elements of an ArrayList of any type that extends Number. As you know, Integer and Double are subclasses of Number, but there are others, such as Short, Float, and even Math.BigInteger, which uses a List of some sort to hold an integer of any size.

```
/** Return the sum of b as a double */
public static double sum(
            ArrayList<? extends Number> b) {
   double s = 0.0;
   for (Number n : b)
      s= s + n.doubleValue();
   return s;
}
```

Important points to note about this method are the following.

1. A foreach-loop is used to process the numbers in the ArrayList. Note that variable n is of type Number, the upper bound on the wildcard.

2. Each subclass of Number has a function doubleValue, which extracts the value in an object as a double value.

Function max

Interface Comparable<T> defines function compareTo. A call ob1.compareTo(ob2) is supposed to return -1, 0, or 1 depending on whether ob1 is smaller, equal to, or greater than ob2.

Generic function max, shown to the right, returns the maximum of three values for any class that implements Comparable. It illustrates two important points:

```
/** Return the maximum of x, y, and z. */
public static <T extends Comparable<T>>
                               T max(T x, T y, T z) {
   if (y.compareTo(x) > 0) x= y;
   // x is the largest of the original x, y
   if (z.compareTo(x) > 0)  return z;
   return x;
}
```

1. Comparable is an interface. In this context, Java uses keyword “extends” for both classes and interfaces.

2. Not only a wildcard but also a type parameter can be used with “extends” and “super”. Here, the declaration of T is

   <T extends Comparable<T>>

   It restricts T to classes that implement Comparable<T>.

Note that T is the type of each parameter and also the return type.

The body of function max illustrates the simplest and most efficient way to determine the maximum of three values. It has only two comparisons, and at most one assignment is executed.

Here are two possible calls of max; in the first autoboxing is used to wrap the ints in Integer objects.

```
max(1, 5, 3)                    returns 5
max("abc", "c", "bdd")          returns "c"
```

Ordering components of a Pair

Recall class Pair, shown to the right. We write a function that swaps fields first and second of a Pair, if necessary, to put the larger first —provided that (1) fields first and second have the same type and (2) the type implements interface Comparable.

```
/** Put largest component of p first. */
public static <T extends Comparable<T>> void
                              order(Pair<T, T> p) {
   if (p.second.compareTo(p.first) > 0) {
      T t= p.second; p.second= p.first;  p.first= t;
   }
}
```

```
/** An instance contains an ordered pair. */
public class Pair<E, F> {
   public E first;      // First element
   public F second; // Second element

/** Constructor: a pair e, f */
   public Pair(E e, F f) {
      first= e;    second= f;
   }

   /** return a representation of this pair. */
   public @Override String toString() {
      return "(" + first + ", " + second + ")";
   }
}
```