Generic Types

In code written before Java 1.5, all items put into a Collection were reduced to type Object. When you accessed the item, you had to remember to cast the Object explicitly. There was always the danger that the programmer would inadvertently put in an object of the wrong type which later would break the cast. For example,

List myList = new ArrayList(); //old style, non-generic

myList.add(″TJHSST″);

myList.add(1.0);

String str0 = (String)myList.get(0); //cast

String str1 = (String)myList.get(1); //throws ClassCastException

(If you remember, our ListNode stored Objects, which meant you had to cast when you accessed the items. In other words, ListNode is not generic. Neither is TreeNode.)

Since Java now includes *generic types*, we can write a generic class and provide the specific type later, in the driver, during instantiation. The compiler performs the necessary type checking on the generic type and guarantees that a ClassCastException for the specific type, whatever it is, is never thrown at runtime. The generic code is thus type-safe.

Java uses the following naming conventions for the formal type parameter:

* <E> for an Element of a Collection
* <T> for a Type parameter
* <K,V> for Key and Value, which are used in maps
* <N> for Number, which is the superclass of Integer and Double
* <?> for any unknown Object type*.* The <?> is known as a wildcard.

In practical terms, if the API shows <E> in the header, e.g. ArrayList<E>, you must supply a concrete class in place of the <E> when you instantiate, e.g. ArrayList<String> or ArrayList<Widget> or any other concrete class. The ArrayList now “knows” the type of the element and automatically supplies any casts as necessary. For example,

List<String> myList = new ArrayList<String>(); //generic type

myList.add(″TJHSST″);

myList.add(″Colonials″);

String str0 = myList.get(0); //automatic casting

String str1 = myList.get(1);

You can read more about generic type declarations here:

<https://docs.oracle.com/javase/tutorial/extra/generics/index.html>

(turn over . . .)

Generic Methods

Java (beginning in Java 1.5) supports **generic methods** that allows the method to work with any given object (or type). You don’t need to cast in your code, nor do you need to write a different version of the method to work with each different object. A generic method specifies the **type parameter** <T> in the header, placed before the return type. The type parameter <T> defines dependencies among the types of the arguments to the method and/or its return type. The compiler guarantees that the dependencies among the generic types are satisfied, and therefore are *type-safe*.

In today’s lab you will see this **generic method** header

public static <T extends Comparable<T>> void sort(List<T> array)

which defines a dependency which limits the kind of T that is allowed in List<T>. It is saying that the objects in List<T> must be part of a hierarchy that somewhere implements the Comparable<T> interface. That dependency makes sense, because, as you know, in order to sort objects, the object T must possess a compareTo method.

This header

public static <T> void swap(List<T> array, int a, int b)

says that objects of type T in a List, whatever they actually are, can be swapped with this method.

If there are no dependencies, we can use the wildcard <?> notation, meaning any Object, as in

public static void output(List<?> array)

Correctly writing generic methods can be complicated. In today's lab the generic type headers are given to you. You will write the bodies of the methods.

If you are interested, read more about generic methods at

<https://docs.oracle.com/javase/tutorial/extra/generics/methods.html>

**Assignment**: The methods in this lab are **generic** **methods**. The headers are given to you. You are to write sort generically, so that the method sorts any concrete type you supply at run-time, first Widgets and then Strings. Use the Selection Sort algorithm.

Notice the <T> dependencies. Notice one of the methods uses a wildcard <?>.

The data is in two files, widgets.txt and strings.txt. The shell is called SortingGenerically.java